НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО

Факультет інформатики та обчислювальної техніки
(назва факультету, інституту)

Кафедра автоматизованих систем обробки інформації і управління
(назва кафедри)

"На правах рукопису"
УДК *621.391*

«До захисту допущено»
Завідувач кафедри

_____ О.А.Павлов
(підпис) (ініціали, прізвище)

“____” _____ 20*18* р.

МАГІСТЕРСЬКА ДИСЕРТАЦІЯ

на здобуття ступеня магістра

за спеціальністю *122 Комп'ютерні науки та інформаційні технології*
(код та назва спеціальності)

спеціалізацією *Інформаційні управляючі системи та технології*
(код та назва спеціалізації)

на тему: *ШВИДКІ АЛГОРИТМИ ОБЧИСЛЮВАЛЬНИХ КРИПТОПРИМІТИВІВ*

Виконав: студент *VI* курсу групи _____
(шифр групи)

Ярушевський Олександр Олегович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник *проф., д.ф.-м.н., проф. Задірака В.К.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант *к.т.н., доц. Жданова О.Г.* _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018

РЕФЕРАТ

Магістерська дисертація: 83 с., 15 рис., 22 табл., 1 додаток, 27 джерел.

Актуальність. Широке впровадження інформаційних технологій робить закономірною та актуальною проблему захисту інформації. Дослідження показують, що лишеполовина фахівців з інформаційної безпеки вважають свою компанію чи установу такою, що готова протистояти сучасним інформаційним загрозам, зокрема і таким, що можуть призвести до неконтрольованого поширення інформації за межі інформаційних систем, у яких вона обробляється

Зв'язок роботи з науковими програмами, планами, темами. Однією з важливих галузей досліджень в системах, мережах і пристроях ІТ є дослідження та розробка нових методів захисту інформації та забезпечення інформаційної безпеки систем, мереж і пристроїв. Захист інформації значною мірою базується на використанні криптографічних методів, пов'язаних з шифруванням даних. У зв’язку з цим удосконалення існуючих методів шифрування та дешифрування є актуальною, що дозволить компаніям та установам підвищити надійсність зашифрованої інформації (підвищити криптостійкість), підвищити безпеку обміну інформації.

Робота виконана на філії кафедри автоматизованих систем обробки інформації та управління в Інституті кібернетики ім. В.М. Глушкова НАН України в рамках науково-дослідної теми «Розробити оптимальні за точністю та швидкодією алгоритми розв’язання задач: інтегрування швидкоосцилюючих функцій, цифрової обробки сигналів та зображень, дистанційного моніторингу об’єктів, інформаційної безпеки» (В.Ф. 140.14, номер державної реєстрації: 0114U000357).

Мета підвищити швидкість шифрування та дешифрування інформації

Для досягнення мети необхідно виконати наступні **завдання**:

– виконати огляд існуючих методів та засобів шифрування та дешифрування інформації;

– здійснити порівняльний аналіз різних алгоритмів шифрування та дешифрування інформації;

– розробити алгоритми шифрування та дешифрування на основі існуючих рішень з використанням методу швидкого обчислення багаторозрядних чисел;

– розробити програмну реалізацію розробленого алгоритму;

– виконати аналіз отриманих результатів.

Об’єкт дослідження – процес шифрування та дешифрування інформації.

Предмет дослідження – алгоритми шифрування та дешифрування інформації, методи швидкого обчислення багаторозрядних чисел.

Наукова новизна одержаних результатів полягає у використанні швидких методів обчислення багаторозрядних чисел для шифрування та дешифрування інформації, що дозволить прискорити існуючі алгоритми шифрування та дешифрування інформації.

Публікації. Матеріали роботи опубліковані в Міжнародної наукової конференції “iScience” та в Міжнародній конференції “ΛΌГОΣ”.

ОБЧИСЛЕННЯ БАГАТОРОЗРЯДНИХ ЧИСЕЛ, АЛГОРИТМИ ШИФРУВАННЯ ТА ДЕШИФРУВАННЯ ІНФОРМАЦІЇ, КРИПТОСТІЙКІСТЬ

ABSTRACT

Master's thesis: 83 pp., 15 fig., 22 tab., 1 app., 27 sources.

The relevance. Wide implementation of information technology makes a logical and topical problem of information security. Studies show that only half of the information security specialists consider their company or institution to be ready to withstand modern information threats, including those that can lead to uncontrolled dissemination of information beyond the information systems in which it is processed. One of the important areas of research in IT systems, networks and devices is the research and development of new methods for protecting information and providing information security for systems, networks and devices. The protection of information is largely based on the use of cryptographic methods related to data encryption. In this regard, the improvement of existing encryption and decryption methods is relevant, which will allow companies and institutions to increase the validity of encrypted information (increase cryptographic stability), increase the security of information exchange.

Relationship with academic programs, plans, themes. Master's thesis is executed according to plan in processes managed optimization department at Institute of Cybernetics of V.M. Glushkov NAS of Ukraine within the research theme «To develop algorithms optimal for accuracy and speed of solving problems: integration of fast-sensing functions, digital processing of signals and images, remote monitoring of objects, information security» (state registration number 0114U000357).

The purpose and objectives of the study. The aim is to increase the speed of encryption and decryption of information

To achieve the aim, we must accomplish the following **tasks**:

– perform an overview of existing methods of encryption and decryption of information;

– perform a comparative analysis of various algorithms for encryption and decryption of information;

– develop algorithms of encryption and decryption based on existing solutions using the method of rapid calculation of multi-digit numbers;

- develop software implementation of the developed algorithm;
- perform the analysis of the obtained results.

The object of research is the process of encryption and decryption of information.

The subject of research - algorithms of encryption and decryption of information, methods of fast calculation of multi-digit numbers.

The scientific novelty of the results is to use fast methods for calculating multi-digit numbers for encryption and decryption of information, which will accelerate existing algorithms for encryption and decryption of information**.**

Publications. The materials of the work are published in the International scientific conference "iScience" and in the International conference “ΛΌΓΟΣ”.

FAST CALCULATION OF MULTI-DIGIT NUMBERS, ALGORITHMS FOR ENCRYPTION AND DECRYPTION INFORMATION, CRYPTOGRAPHY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

FFT – Fast Fourier Transform (Швидке перетворення Фур’є)

ШПФ – Швидке перетворення Фур’є

CLNW – Constant Length Nonzero Window (Ненульові вікна однакової довжини)

VLNW – Variable Length Nonzero Window (Ненульові вікна різної довжини)

ВСТУП

У сучасному інформаційному суспільстві велику загрозу конфіденційності та цілісності інформації представляє кіберзлочинність. Зростання кількості кібератак та доступність програмно-технічних засобів для їх реалізації зумовлює необхідність розробки сучасних засобів інформаційної безпеки громадян та держави в цілому.

Отже, актуальною задачею є створення та вдосконалення систем захисту інформації, зокрема алгоритмів криптографічного захисту, що в більшості випадків є базовим ядром таких систем. На даний час основним питанням, що підлягає вирішенню, є збільшення об'ємів інформації, що може оброблятися функціями криптографічного перетворення. Одним з підходів вирішення даної проблеми є застосування швидких алгоритмів обчислення криптопримітивів.

В основу покладені алгоритми виконання різних операцій над багаторозрядними числами. Для простоти викладення припустимо, що ми маємо справу з цілими числами. Розглянуті алгоритми можна легко розповсюдити на випадок чисел з плаваючою комою тощо.

Під n розрядним цілим числом будемо розуміти довільне ціле число, менше за b^n , де b — основа прийнятої позиційної системи, в якій ми представляємо числа; такі числа в цій системі записуються з використанням не більше ніж n розрядів.

Ці числа можна розглядати як числа, записані в системі числення за основою b, де b — розмір машинного слова. Наприклад, ціле число, яке займає 10 машинних слів в пам'яті ЕОМ, розмір слова якої дорівнює $b = 10^{10}$, має 100 десяткових (цифр; але ми будемо розглядати його як десятирозрядне число за основою 10^{10}. Це робиться шляхом групування бітів.

Далі багаторозрядні цілі числа будемо називати ще багатослівними або s слівними, оскільки кожне з них можна представити у вигляді масиву 16- бітових, 32-бітових або 64- бітових слів.

На теперішній час досліджені алгоритми для наступних операцій:

– додавання або віднімання k-розрядних цілих чисел, з отриманням kрозрядної відповіді та цифри переносу;

– множення k-розрядного цілого числа на m-розрядне ціле число, з отриманням (m+k)-розрядної відповіді;

– піднесення k-розрядного числа до m-розрядного степеня;

– ділення (m+k)-розрядного цілого числа на m-розрядне ціле число з отриманням (m+1)-розрядної частки та k-розрядного лишку;

– виконання модулярних операцій.

Основна направленість дослідження - побудова ефективних за часом реалізації алгоритмів багаторозрядної арифметики. Як резерв оптимізації обчислень використовуються не тільки алгоритмічні новинки і різні моделі обчислень, але й нові архітектурні рішення, що дозволить побудувати ефективні за швидкодією алгоритми розв’язання задач інформаційної безпеки, а саме: підвищення продуктивності систем двоключової криптографії.

1 ОГЛЯД СУЧАСНОЇ КРИПТОГРАФІЇ ТА ЇЇ ПРОБЛЕМАТИКИ

Криптографія – це практика і вивчення методів безпечного спілкування в присутності третіх осіб (так званих противників). У більш загальному понятті, мова йде про побудову та аналізу протоколів, які дозволяють подолати вплив противників і які пов’язані з різними аспектами в області інформаційної безпеки, таких як конфіденційність даних, цілісність даних, аутентифікації і безвідмовності. Сучасна криптографія перетинає дисципліни математики, інформатики та електротехніки. Застосування криптографії включають наступне: банківські карти, комп’ютерні паролі і електронну комерцію, та інше.

В криптографичній термінології вихідне повідомлення називають відкритим текстом (plaintext або cleartext). Зміна вихідного тексту так, щоб скрити від інших його склад, називають шифруванням (encryption). Зашифроване повідомлення називають шифротекстом (ciphertext). Процес, при якому із шифротексту видаляється відкритий текст називають дешифруванням (decryption). Переажно в процесі шифрування і дешифрування використовується деякий ключ (key) і алгоритм забезпечує, що дешифрування можна зробити лише знаючи ключ.

Криптографія – це наука про те, як забезпечити секретність повідомлення. Криптоаналіз – це наука про те, як відкрити шифроване повідомлення, так як відкрити текст не знаючи ключ. Криптографією займаються криптографи, а криптоаналізом займаються криптоаналітики. [6]

Основна класифікація криптографічних систем:

Перш ніж приступитися до класифікації криптологічних систем слід зробити деякі зауваження. По етапах розвитку можна виділити три періоди розвитку:

– перший етап – етап донаукової криптології (до 1949 р.);

– другий етап – етап наукової криптології із секретними ключами (з 1949 р. по сімдесяті роки);

– третій етап – етап наукової криптології з використанням ЕОМ (із сімдесятих по теперішній час).

Звичайно, такий розподіл достатньо умовний, але воно враховує основні події, що вплинули на розвиток криптології як науки. Якщо в стародавньому світі криптологією займалися як мистецтвом, то зараз криптологія стала наукою. Прорив у криптоаналізі був зроблений з моменту виникнення ЕОМ. Слід зазначити, що перші досліди такого роду були початі під час другої світової війни. В Англії в 1942 році вступили в стрій кілька спеціалізованих для злому шифрів ЕОМ, спеціально створених для цього Аланом Тюрінгом. Це була перша у світі досить швидкодіюча ЕОМ за назвою “Колос”. За допомогою цієї машини англійські криптоаналітики менше ніж за день могли розколоти будь-яку шифровку німецької шифрувальної машини Энігма.

Створення персональних ЕОМ відкрило нову сторінку в криптології. З однієї сторони різко зросли можливості криптоаналітиків. Це в першу чергу стосується криптоаналізу за допомогою простого перебору. З іншої сторони з’явились практично необмежені можливості у шифрувальників, які, використовуючи можливості ПЕОМ, створювати шифри, що практично не розкриваються.

На першому етапі розвитку криптології існувало два основних типу перетворень відкритих текстів – заміни й перестановки. Шифрів перестановки відомо достатньо велика кількість – це й шифр «скітала», шифруючи таблиці й ін. Головна ідея шифрів перестановки є заміна місця розташування символів відкритого тексту. Шифрів заміни було значне більше, але всі вони будувалися на заміні символу відкритого тексту символом зашифрованого тексту. До таких шифрів належать полібіанський квадрат, шифруючи таблиці, Трисемуса, система шифрування Віжінера й ін.

На другому етапі передбачалося, що для шифрування й розшифрування використовується один секретний ключ. Дані системи були названі симетричними. Як було сказано вище, основними принципами шифрування стають розсіювання й перемішування. У міру розвитку криптологія в

симетричних криптологічгих системах виділяються два головних напрямки шифрування: блокові й потокові шифри. У блокових шифрах відкритий текст розбивається на блоки фіксованої довжини й зазнає шифруванню. Причому кожний блок зашифровується своїм шифром, але алгоритм перемішування залишався однаковим для всіх блоків. На цьому принципі побудовані велика кількість шифрів, включаючи американський стандарт DES і національний стандарт ДЕРЖСТАНДАРТ 28147-89. У блокових шифрах широко використовується перемішування. При використанні потокових шифрів кожний символ відкритого тексту зашифрується незалежно від інших. Головною проблемою створення потокового шифру є створення послідовності, що шифрує. Вимоги до таких послідовностей досить тверді. Як тільки ми починаємо говорити про шифри, перед нами встає проблема їх передачі до одержувача. При використанні потокових шифрів вони можуть вироблятися як на передавальному, так і на прийомному кінцях лінії зв’язку.

Слід зазначити той факт, що в цей час практично жоден шифр не є чистим, у тому розумінні, що ставиться до одному з видів. Найчастіше використовується комбінація декількох шифрів.

Наступним більшим класом є асиметричні криптологічні системи або системи з відкритим ключем. Головною ідеєю при створенні цього класу шифрів є генерація двох ключів. Один відкритий ключ поширюється по відкритих каналах зв’язку й використовується при шифруванні повідомлень. На прийомній стороні за допомогою секретного ключа проводиться розшифрування повідомлення. Основою при створенні таких шифрів, як сказано вище, є задачі з важким розв’язком.. У якості таких задач у цей час використовуються задачі факторизації, дискретного логарифмування й методи теорії завадостійкого кодування. [6]

Класифікація алгоритмів шифрування представлено на рис 1.1.

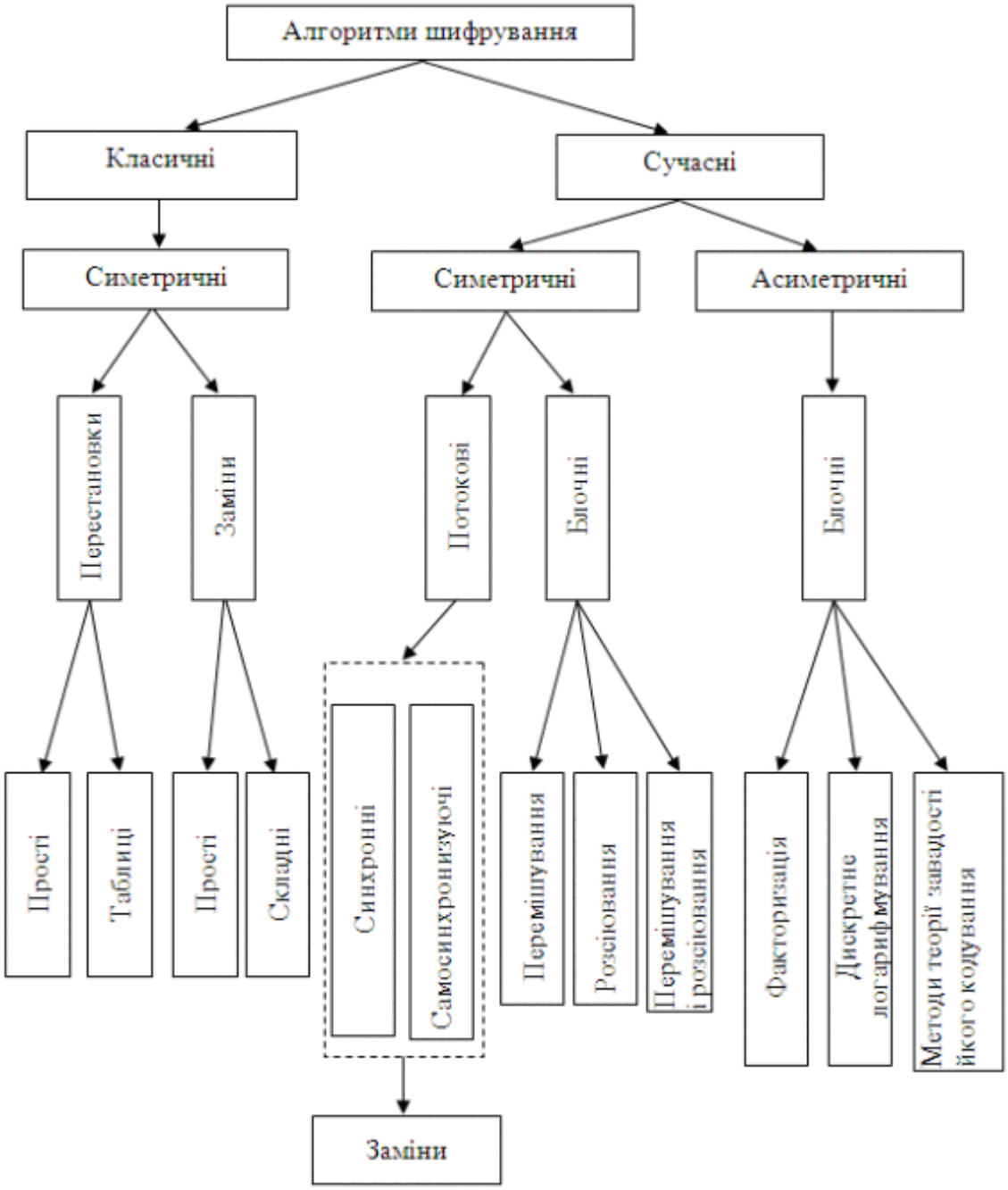

Рисунок 1.1 – Класифікація алгоритмів шифрування

Асиметричні алгоритми шифрування – алгоритми шифрування, які використовують різні ключі для шифрування та розшифрування даних. Головне досягнення асиметричного шифрування в тому , що воно дозволяє людям, що не мають існуючої домовленості про безпеку, обмінюватися секретними повідомленнями. Необхідність відправникові й одержувачеві погоджувати таємний ключ по спеціальному захищеному каналі цілком відпала. Процедура

шифрування обрана так, що вона необоротна навіть по відомому ключу шифрування. Тобто, знаючи ключ шифрування й зашифрований текст, неможливо відновити вихідне повідомлення – прочитати його можна тільки за допомогою другого ключа – ключа дешифрування. А раз так, то ключ шифрування для відправлення листів якій-небудь особі можна взагалі не приховувати – знаючи його однаково неможливо прочитати зашифроване повідомлення. Тому, ключ шифрування називають в асиметричних системах "відкритим ключем", а от ключ дешифрування одержувачеві повідомлень необхідно тримати в секреті – він називається "закритим ключем". Алгоритми шифрування й дешифрування створюються так, щоб знаючи відкритий ключ, неможливо було обчислити закритий ключ. [6]

Симетричні алгоритми шифрування – спосіб шифрування, в якому для шифрування і дешифрування застосовується один і той же криптографічний ключ. До винаходу схеми асиметричного шифрування єдиним існуючим способом було симетричне шифрування. Ключ алгоритму повинен зберігатися в секреті обома сторонами. Алгоритми шифрування і дешифрування даних широко застосовуються в комп'ютерній техніці в системах приховування конфіденційної і комерційної інформації від не коректного використання сторонніми особами. Головним принципом у них є умова, що та приймає заздалегідь знають алгоритм шифрування, а також ключ до повідомлення, без яких інформація є всього лише набір символів, що не мають сенсу. Симетричні криптоалгоритми виконують перетворення невеликого (1 біт або 32-128 біт) блоку даних в залежності від ключа таким чином, що прочитати оригінал повідомлення можна тільки знаючи цей секретний ключ.

Симетричні криптоалгоритми діляться на:

– потоковими;
– блокові шифри.

Потоковими називаються програмні або апаратні реалізації алгоритму, що дозволяють шифрувати побітно безперервні потоки інформації. Сам скремблер представляє із себе набір бітів, що змінюються на кожному кроці по певному

алгоритму. Після виконання кожного чергового кроку на його виході з'являється біт, що шифрує, – або 0, або 1, що накладається на поточний біт інформаційного потоку операцією XOR («побітне виключаюче АБО»). Основним недоліком алгоритмів скремблювання є їхня нестійкість до фальсифікації.

Блокові шифри в ході своєї роботи роблять перетворення блоку вхідної інформації фіксованої довжини і одержують результуючий блок того ж обсягу, але недоступний для прочитання стороннім особам, що не володіють ключем. Таким чином, схему роботи блокового шифру можна описати функціями Z = EnCrypt(X,Key) і X = DeCrypt(Z,Key). Ключ Key є параметром блокового алгоритму і являє собою деякий блок двійкової інформації фіксованого розміру. Вихідний (X) і зашифрований (Z) блоки даних також мають фіксовану розрядність рівну між собою, але необов'язково рівну довжині ключа.

Порівняння асиметричних і симетричних криптоалгоритмів.

Асиметричні криптоалгоритми:

– криптосистема з відкритим ключем;

– для шифрування повідомлення використовується відкритий ключ, а при дешифруванні – закритий, тобто, знаючи ключ шифрування й зашифрований текст, неможливо відновити вихідне повідомлення;

– при порушенні конфіденційності k-ої робочої станції зловмисник довідається тільки "закритий" ключ k, це дозволяє йому читати всі повідомлення, що приходять абонентові k, але не дозволяє видавати себе за нього при відправленні листів;

– в асиметричних системах кількість існуючих ключів пов'язане з кількістю абонентів лінійно (у системі з N користувачів використовуються 2*N ключів).

Симетричні криптоалгоритми:

– криптосистема з секретним ключем;

– секретний ключ використовується і для шифрування, і для дешифрування, тобто, знаючи ключ шифрування й зашифрований текст, ви зможете дешифрувати повідомлення;

– при порушенні конфіденційності якої-небудь робочої станції зловмисник одержує доступ до всіх ключів цього користувача й може відправляти, нібито від його імені, повідомлення всім абонентам, з якими “жертва” вела переписку;

– в симетричних системах число ключів зростає квадратично із збільшенням числа користувачів. [6]

Алгоритм Діффі – Хеллмана. Одна з фундаментальних проблем криптографії — безпечне спілкування по прослуховуючому каналу. Повідомлення потрібно зашифровувати і розшифровувати, але для цього обом сторонам потрібно мати загальний ключ. Якщо цей ключ передавати по тому ж каналу, то прослуховуюча сторона теж отримає його, і сенс шифрування зникне.

Алгоритм Діффі — Хеллмана дозволяє двом сторонам отримати загальний секретний ключ, використовуючи незахищений від прослуховування, але захищений від підміни канал зв’язку. Отриманий ключ можна використовувати для обміну повідомленнями за допомогою симетричного шифрування.

Нехай n деяке велике ціле число й нехай g інше ціле, *92* що лежить строго між *1* і $n-1$. У якості першої дії протоколу Діффі-Хеллмана Аліса й Боб домовляються про параметри *n* й *g* за допомогою несекретного каналу зв’язку (альтернативно *n*й *g* могли б бути стандартними параметрами, застосовуваними всіма користувачами системи).

Потім Аліса вибирає деяке велике ціле число *x* й обчислює $X = g^x \pmod n$. Відповідно, Боб вибирає y й обчислює $Y = g^y \pmod n$. Після цього Аліса й Боб обмінюються *X* й *Y* по одному несекретному каналу зв’язку, зберігаючи при цьому в секреті *x* й *y* (*x* знає тільки Аліса, а *y* – тільки Боб). Нарешті, Аліса обчислює $Y^x \pmod n$, відповідно, Боб обчислює $X^y \pmod n$. Обоє ці значення рівні між собою, тому що кожне з них дорівнює $g^{xy} \pmod n$. Це і є саме той ключ *k* , який вони хотіли спільно сгенерувати. [22]

RSA алгоритм. RSA – це криптографічний алгоритм з відкритим ключем, що грунтується на обчислювальній складності задачі факторизації великих цілих чисел.

Криптосистема RSA стала першою системою, придатної і для шифрування, і для цифрового підпису.

Алгоритм створення відкритого і секретного ключів

Крок 1. Вибираються два різних випадкових простих числа p і q.

Крок 2. Обчислюється їх добуток $n=pq$, який називається модулем.

Крок 3. Обчислюється значення функції Ейлера від числа n:

$$\varphi(n) = (p-1)(q-1).$$

Крок 4. Вибирається ціле число e $(1 < e < \varphi(n))$, взаємно просте з значенням функції $\varphi(n)$.

Крок 5. Обчислюється число d, мультиплікативно зворотне до числа e по модулю $\varphi(n)$, тобто число, що задовольняє умові: $de \equiv 1 \; mod \; \varphi(n)$;

Крок 6. Пара *{e, n}* публікується в якості відкритого ключа RSA;

Крок 7. Пара *{d, n}* відіграє роль закритого ключа RSA і тримається в секреті.

Алгоритм шифрування

Крок 1. Взяти відкритий ключ (*e, n*) Сергея;

Крок 2. Взяти відкритий текст m;

Крок 3. Зашифрувати повідомлення з використанням відкритого ключа Сергея: $c = E(m) = m^e \; mod \; n$;

Алгоритм дешифрування

Крок 1. Прийняти зашифроване повідомлення C;

Крок 2. Взяти свій закритий ключ (*d, n*);

Крок 3. Застосувати закритий ключ для розшифрування повідомлення: $m = D(c) = c^d \; mod \; n$ [19]

При практичній реалізації відомих методів асиметричної криптографії, які базуються на методах ключів загального доступу, необхідно використовувати ефективні за швидкодією алгоритми виконання арифметичних операцій над великими числами. До таких операцій належать операції додавання, віднімання, множення, ділення, піднесення до степеня і обчислення лишку багаторозрядних цілих чисел, тобто чисел, розрядність яких (n бітів) перевищує довжину

розрядного слова. Такі числа будемо називати ще багатослівними або s-слівними (тому що кожне з них можна представити у вигляді масиву з 16-, 24-, 32- або 64-бітових слів на PC AT). Якщо n велике (n > 1024 біт), то виконання таких операцій вимагає істотних витрат машинного часу і виникає необхідність в оптимізації за швидкодією відповідних алгоритмів і програм. Дослідженню цих питань приділяється досить велика увага [14, 16, 20, 25].

Розглянемо час виконання операції множення багаторозрядних чисел на процесорі з фіксованою довжиною слова пропорційний квадрату довжини операнда $O(k^2)$. На сьогоднішній день існують методи, які дають змогу обчислити добуток швидше. Це, наприклад, метод Карацуби-Офмана [2], складність якого дорівнює $O(k^{\log_2 3})$, де $\log_2 3 \approx 1.58$, та метод многочленів з часом виконання порядку k 1+ε , $\varepsilon > 0$. Алгоритм Шенхаге-Штрассена [3] дає змогу помножити два k-розрядних числа за $O(k \log k \log\log k)$ кроків. Ці методи базуються на зведенні множення n-розрядних чисел до множення чисел з меншим числом розрядів.

С. Кук разом С. Ондеро довели теорему про найкращу нижню границю, в яків стверджується, що при певних обмеженнях не існує алгоритму, який би множив n-розрядні числа менше ніж за $O(\frac{k \log k}{(\log\log k)^2})$ операцій.

Використовуючи алгоритм для швидкого множення, можна показати, що ділення також можна виконати за $O(k \log k \log\log k)$ циклів.

Якщо ж спеціально для виконання операції множення сконструювати послідовно-паралельний апаратний пристрій, то час може бути зменшений до $O(k)$. У цьому разі потрібна кількість логічних елементів буде пропорційна довжині операндів $O(k)$. Час роботи самих швидкодіючих реалізацій пропорційний $O(\log k)$, але вони потребують порядку $O(k^2)$ логічних елементів [1].

При шифруванні, розшифруванні, керуванні ключами, у криптологічних протоколах, високоточних обчисленнях використовується операція піднесення до степеня, яка є найбільш складною. Причому операція піднесення до степеня

реалізується через операції множення та піднесення до квадрата. Оскільки операція множення є ключовою операцією при піднесенні до степеня і, отже, на неї лягає основне навантаження в алгоритмах асиметричної криптографії основна увага буде приділена алгоритмам виконання операції множення, аналізу й тестуванню оцінок їхньої якості [1].

Запропоновано оригінальні підходи до оггтимізації за швидкодією алгоритмів множення багато розрядних цілих чисел і відповідних програм. Розглядаються два підходи до оптимізації за часом виконання на комп'ютері операції множення: один - заснований на залученні деяких резервів оптимізації алгоритмів, інший - на розробці більш ефективної структури програми, що реалізує алгоритм.

Як показує аналіз складності використовуваних алгоритмів множення великих чисел, при розробці криптографічних методів захисту інформації з відкритим ключем необхідно мати бібліотеку програм. які реалізують різні алгоритми швидкого множення, кожний з яких має свою область ефективного застосування залежно від значень n (довжини слова), моделі обчислень, апаратної або програмної реалізації. Це дозволяє оптимізувати обчислювальні витрати, необхідні для реалізації алгоритмів асиметричної криптографії.

Відомо, що стандартний алгоритм множення вимагає $O(n^2)$ операцій для множення двох n-розрядних чисел. Більш якісні алгоритми можна одержати, використовуючи ідею Карацуби-Офмана та її рекурентного застосування, а також швидкі дискретні ортогональні перетворення [5, 10, 22, 27]. Таким чином, серед найбільш швидких алгоритмів множення великих чисел можна виділити алгоритм Карацуби-Офмана [8, 17, 20], який потребує $O(n^{log_2 3})$, де $log_2 3 \approx 1.58$, метод многочленів з часом виконання n 1+ε , $\varepsilon > 0$, алгоритм Тоома—Кука [21, 24] зі складністю порядку$O(2n^{\sqrt{2log_2 n}}\ log_2 n)$, медулярний метод, заснований на ідеях медулярної арифметики [75], алгоритм Шенхаге—Штрасена [14, 20, 22, 26], який потребує $O(nlog_2 n)$ операцій. Суттєвого зменшити час виконання операції множення дозволяє використання алгоритмів обчислення добутку n-розрядних чисел за модулем за допомогою добутку Монтгомері [4, 5, 21] та

розробка швидких алгоритмів обчислення арифметичної згортки на основі не тільки алгоритму швидкого перетворення Фур'є (ШПФ), а й швидких лінійних дискретних перетворень, наприклад, Уолша і Хаара [15, 17], а отримати більш якісну реалізацію - максимальне використання можливості обчислювальної техніки, мов програмування й т.д.

Усі ці алгоритми швидкого множення базуються на ідеї зведення множення n-розрядних чисел до множення чисел з меншою кількістю розрядів.

1.1 Ефективні за складністю алгоритми виконання операцій над багаторозрядними числами

Алгоритм Карацуби-Офмана та його оптимізація

У роботі [6] в якості базового алгоритму множення (тобто такого, з яким порівнюються інші алгоритми) вибрано алгоритм, що реалізує класичну прямокутну схему, а в роботі [7] - алгоритм, що реалізує діагональну схему. Він порівняно з прямокутною схемою вимагає меншого числа звернень до пам'яті ЕОМ при формуванні результату (при множенні s-слівного числа на m-слівне, прямокутна схема вимагає $2sm$ звернень, а діагональна - 2(m+s-1)).

Реалізація діагональної схеми в програмі під назвою MD (множення діагональне) призначена для множення двох s-слівних чисел. Ця програма містить багато команд умовного переходу, оскільки повинна слідкувати за довжиною кожної діагоналі і їх кількістю. Якщо s фіксоване, то команди умовного переходу можна виключити, а також використати більш ефективні способи адресації команд. Будемо називати це "розшивкою" програми. Однак з ростом s довжина відповідних програм суттєво збільшується. Це видно з таблиці 1.1, в якій наведено довжини програм MDR (діагональна схема з "розшивкою") в кілобайтах для низки значень 5. У цій таблиці для $s < 13$ наведено реальні розміри довжин $L_{об}(s)$) об'єктних модулів, а для решти s - обчислені їх оцінки за формулою (формула для $L_{об}(s)$ отримана шляхом аналізу програм для різних значень s і виявлення закономірностей зміни їх довжин від зміни s)

$$L_{об}(s) = \frac{s^2 + 2s}{80}$$

Таблиця 1.1 – Залежність довжини програми від s-слівних чисел

s	4	5	7	8	12	13	30	52	64
$L_{об}(s)$	0.3	0.4	0.8	1.0	2.1	2.4	12.0	35.1	52.8

Для порівняння зазначимо, що програма MD займає приблизно 1 кілобайт пам'яті.

У таблиці 1.2 для деяких значень *s* наведено оцінки коефіцієнтів прискорення часу роботи програми MDR відносно програми MD.

Таблиця 1.2 – Оцінки коефіцієнтів прискорення часу роботи

s	4	5	7	8	12	13
K(MDR, MD)	0.54	0.54	0.56	0.56	0.57	0.57

Проаналізувати вміст цих двох таблиць і зробити відповіді висновки нескладно.

Побудова ефективніших за часом реалізації алгоритмії множення базується на зменшенні загального числа однослівних множень. Розглянемо одну з таких процедур, яка описана в [5] і з спрощенням процедури, запропонованої в роботі [6].

Нехай U_{2s} і V_{2s} – цілі числа, кожне з яких розміщене в 2s 16 бітових словах. Якщо подати їх у вигляді $U_{2s} = U_s^1 2^{16s} + U_s^0$, $V_{2s} = V_s^1 2^{16s} + V_s^0$ і позначити $X = 2^{16s}$, то добуток можна записати так:

$$U_{2s}V_{2s} = (U_s^1 X + U_s^0)(V_s^1 X + V_s^0) == U_s^1 V_s^1 X + [U_s^1 V_s^1 + U_s^0 V_s^0 - (U_s^1 - U_s^0)(V_s^1 - V_s^0)]X + U_s^0 V_s^0 \quad (1.2)$$

Таким чином, задачу множення 2s-слівних чисел зведено до трьох множень s-слівних чисел, двох віднімань s-слівних і трьох додавань 2s-слівних.

Отже, якщо позначити через T(s) - час виконання операції множення s-слівних чисел, то застосування формули (1.2) дає

$$T(2s) \leq 3T(s) + C$$

де C – час виконання необхідних при цьому операцій додавання, віднімання.

Прийом множення чисел, який базується на формулі (1.2) будемо називати “розщепленням”.

Алгоритм, який базується на процедурі швидкого перетворення Фур ’є

Метод множення багаторозрядних чисел з використанням швидкого перетворення Фур’є (ШПФ) був запропонований В.Штрассеном у 1973р. [2] і базується на ідеї використання теореми про дискретну згортку двох функцій для множення великих чисел, оскільки добуток багаторозрядних чисел без врахування переносів є дискретною циклічною згорткою.

Це можливе тому, що ціле число можна представити поліномом. А множення поліномів еквівалентно згортці коефіцієнтів многочленів, що перемножуються. Наприклад, $348 = 3x^2 + 4x + 8$ при x = 10. Аналогічно, при x = 10, $8x^2 + 5x + 7 = 857$.

Виконуючи множення 348 на 857, ми можемо спочило перемножити поліноми

$(3x^2 + 4x + 8)(8x^2 + 5x + 7) = 24x^4 + 47x^3 + 105x^2 + 68x + 56$, потім оцінити результуючий поліномом

$24(10^4) + 47(10^3) + 105(10^2) + 68(10) + 56 = 298236$ і отримати добуток $348 \cdot 857 = 298236$.

Отже, якщо ми можемо множити поліноми швидко, то можемо множити швидко і багаторозрядні числа. Для множенні двох поліномів використаємо дискретне перетворення Фур’є((ДПФ) та теорему про дискретну згортку функцій.

Дійсно, нехай $k = 2^y, y > 0$ - ціле, необхідно отримав добуток двох k-розрядних чисел u і v: c = uv – 2k-розрядне двійкове число. Виберемо натуральні числа l і m такі, що

$$l \cdot 2^m \geq 2k$$

Розіб’ємо числа u і v на $M = 2^m$ блоків довжиною l:

$$u = \sum_{j=0}^{2^m-1} u_j 2^{jl},\ v = \sum_{j=0}^{2^m-1} v_j 2^{jl},$$

де $0 \le u_j \le 2^l, 0 \le v_j \le 2^l$ і $u_j = v_j = 0$ для $j \ge 2^{m-1}$.

Тоді їх добуток с має вигляд:

$$c = u \cdot v = \left(\sum_{j=0}^{2^m-1} u_j 2^{jl} \right) \left(\sum_{j=0}^{2^m-1} v_j 2^{jl} \right) = \sum_{r=0}^{2^m-1} c_r 2^d$$

$$c_r = \sum_{\rho+\delta=r(mod\ 2^m)} u_\rho \cdot v_\delta$$

представляє собою циклічну згортку u_ρ з v_δ. Для обчислення c_r доцільно використати теорему про дискретну згортку двох функцій і алгоритм ШПФ з попередньою заготовкою елементів матриці перетворень.

Подамо покроковий опис алгоритму множення uv:

Крок 1. Обчислити за допомогою ШПФ дискертні перетворення Фур'є:

$$\hat{u}_s = \sum_{\rho=0}^{2^m-1} u_\rho W_m^{ps}, \hat{v}_s = \sum_{\delta=0}^{2^m-1} u_\delta W_m^{\delta s},$$

де $W_m = \exp\left(-\frac{2\pi i}{2^m}\right), i = \sqrt{-1}$;

Крок 2. Обчислити $\hat{c}_s = \hat{u}_s \cdot \hat{v}_s$;

Крок 3. Обчислити обернене ДПФ від $\hat{c}_s$ для отримання c_r з використанням алгоритму ШПФ:

$$c_r = \frac{1}{2^m} \sum_{s=0}^{2^m-1} \hat{c}_s \cdot W_m^{-s\tau};$$

Крок 4. Обчислити с, використовуючи зсуви і додавання l-розрядних чисел.

Оскільки, при виконанні ШПФ використовуються комплексні числа W_m^β, а перемножуються цілі числа, необхідно застосувати досить точні наближення для W_m^β. Похибка заокруглення при цьому повинна бути настільки малою, щоб похибки заокруглення цілих чисел c_rбули меншими 1/2 і тим самим точно визначались з заокруглень (якщо c_r знаходиться між двома цілими числами, то результатом заокруглення є найближче ціле число). Для цього необхідно визначити число q - кількість вірних значущих цифр, необхідних для

наближеного обчислення W_m^{β}, і величин, отриманих на кожному кроці алгоритму ШПФ.

2 АЛГОРИТМИ ШВИДКОГО МНОЖЕННЯ ЧИСЕЛ

Як було зазначено у вступі, характерною особливістю розв'язання багатьох задач апроксимації функцій, моделювання фізичних, хімічних (біохімічних) процесів, аеродинаміки, гідродинаміки, захисту інформації є використання обчислень з багатократною точністю або над багаторозрядними числами. Це обумовлює актуальність створення ефективних алгоритмів виконання операцій над багаторозрядними числами для програмної реалізації на універсальних ЕОМ і для спеціалізованних апаратних та програмно-апаратних комплексів.

Алгоритми обчислення степеня за модулем виконуються операції піднесення до квадрату та множення за модулем на кожному кроці піднесення до степеня. Для обчислення $x^n \bmod M$ треба кожний раз застосовувати, програми множення за модулем.

Розглянемо алгоритм обчислення

$$R = a \cdot b \bmod M,$$

де *a, b, M* – *k*-розрядні числа. Оскільки *k* часто більше ніж 256, то необхідно так будувати структури даних, щоб оперувати з дуже великими числами. Припускаючи, що довжина машинного слова дорівнює ω (як правило ω = 16 або 32), розіб'ємо *k*-бітове число на *s* слів так, що $(s-1)\omega < k \le s \cdot \omega$Проміжні результати можуть займати більше ніж *s* слів, і їх також треба зберегти.

2.1 Стандартний алгоритм

Нехай a і b два s-слівні числа, записані за основою *w*

$$a = (a_{s-1}a_{s-2} \dots a_0) = \sum_{i=0}^{s-1} a_i w^i,$$

$$b = (b_{s-1}b_{s-2} \dots b_0) = \sum_{i=0}^{s-1} b_i w^i,$$

де a і b – цифри з проміжку [0; *w*-1]. Взагалі *w* може бути будь-яким цілим числом. Для використання на комп'ютері часто вибирається $w = 2^{\omega}$, ω –

довжина машинного слова, наприклад, $\omega = 32$. Стандартний алгоритм добутку a і b знаходить часткові добутки, підсумовує їх і отримує кінцевий 2s-слівний результат t. Нехай t_{ij} означає пару чисел (Перенос, Сума) – (C, S), як результат добутку $a_i \cdot b_i$. Наприклад, якщо w = 10, a_i = 7, b_i = 8, то $t_{ij} = (5, 6)$. Пари t_{ij} можуть бути розміщені у вигляді таблиці наступним чином:

				a_3	a_2	a_1	a_0
				b_3	b_2	b_1	b_0
				t_{03}	t_{02}	t_{01}	t_{00}
			t_{13}	t_{12}	t_{11}	t_{10}	
		t_{23}	t_{22}	t_{21}	t_{20}		
	t_{33}	t_{32}	t_{31}	t_{30}			
t_7	t_6	t_5	t_4	t_3	t_2	t_1	t_0

Останній рядок означає суму часткових добутків представляє собою 2s-слівний добуток. Стандартний алгоритм по суті цифра за цифрою виконує описане множення і додавання. Для економії пам’яті використовується тільки одна змінна t для часткового добутку. Її початкове значення дорівнює 0; потім береться цифра множника b, множиться на a і підсумовується; для часткового добутку t. Наприкінці обчислень ця змінна для часткових добутків містить остаточний добуток $a \cdot b$.

Стандартний алгоритм обчислення $a \cdot b$ має наступний вигляд:

На вході: a, b

На виході: $t = a \cdot b$

Крок 0. Спочатку $t_i = 0$ для всіх $i = \overline{0, 2s-1}$.

Крок 1. Для $i = \overline{0, s-1}$

Крок 2. $c = 0$

Крок 3. Для $j = \overline{0, s-1}$

Крок 4. $S = t_{i+j} + a_j b_j + C$

Крок 5. $t_{i+j} = S$

Крок 6. $t_{i+S} = C$

Крок 7. Результат $(t_{2s-1}, t_{2s-2}, \dots, t_0)$

В таблиці 2.1 наведені крок обчислення $a \cdot b = 348 \cdot 857$ за допомогою стандартного алгоритму:

Таблиця 2.1 – Покроковий опис алгоритму множення

i	j	Крок	(*C, S*)	Значення *t*
0	0	$t_0 + a_0 b_0 + C$ $0 + 8 \cdot 7 + 0$	(0, *) (5, 6)	000000 000006
	1	$t_1 + a_1 b_0 + C$ $0 + 4 \cdot 7 + 5$	(3, 3)	000036
	2	$t_2 + a_2 b_0 + C$ $0 + 3 \cdot 7 + 3$	(2, 4)	000436
				002436
1	0	$t_1 + a_0 b_1 + C$ $3 + 8 \cdot 5 + 0$	(0, *) (4, 3)	002436
	1	$t_2 + a_1 b_1 + C$ $4 + 4 \cdot 5 + 4$	(2, 8)	002836
	2	$t_3 + a_2 b_1 + C$ $2 + 3 \cdot 5 + 2$	(1, 9)	009836
				019836
2	0	$t_2 + a_0 b_2 + C$ $8 + 8 \cdot 8 + 0$	(0, *) (7, 2)	019236
	1	$t_3 + a_1 b_2 + C$ $9 + 4 \cdot 8 + 7$	(4, 8)	018236
	2	$t_4 + a_2 b_2 + C$ $1 + 3 \cdot 8 + 4$	(2, 9)	098236
				298236

Для застосування цього алгоритму необхідно виконати крок 4:

$$(C, S) = \mathrm{t}_{i+j} + a_j b_j + C,$$

де t_{i+j}, a_j, b_j, C і S – однослівні w-бітові числа. Цей крок називається операцією внутрішнього добутку і зустрічається в багатьох арифметичних та теоретико-числових обчисленнях. Операція внутрішнього добутку, як описано вище, потребує множення двох w-бітових чисел, потім додавання отриманого результату з w- бітовим переносом результату попереднього внутрішнього добутку і нарешті, додавання отриманого результату до поточного слова t_{i+j} часткового добутку. В результаті цих трьох дій отримуємо $2w$-бітове число.

Оскільки крок внутрішнього добутку знаходиться в середині самого внутрішнього циклу, його слід виконувати як можна швидше. Аналіз крюків цього алгоритму показує, що загальне число кроків внутрішнього добутку дорівнює s^2. Так як s=k/ ω, ω – постійне для конкретного комп'ютера, то стандартний алгоритм множення потребує $O(k^2)$ бітовий операцій для множення двох k-бітових чисел. Цей алгоритм асимптотично повільніший, ніж алгоритм Карацуби та алгоритм, який використовує швидке перетворення Фур'є (ШПФ). Однак він простіший для застосування та має кращу швидкодію для малих чисел, ніж ці асимптотично більш швидкі алгоритми.

2.2 Алгоритм швидкого множення цілих чисел, використовуючи швидке перетворення Фур'є

Алгоритм множення цілих чисел, використовуючи швидке перетворення Фур'є дозволяє прискорити множення від $O(n^2)$ до $O(n \cdot log_2(n))$. Застосувати алгоритм швидкого перетворення Фур'є нам дозволяє тей факт, що число можна представити у вигляді поліному. Далі ми можемо перейти від по точкового представлення до ключ-значення і виконаємо по точково множення двох поліномів, після виконавши інтерполяцію, отримаємо результуючий поліном, який вже можна привести до цілого числа. В наступних пунктах більш детально розглянемо даний підхід.

2.2.1 Представлення цілого числа у вигляді поліному

Як зазначили раніше ціле число можна представити у вигляді поліному.

Існує два способи представлення поліномів: з допомогою коефіцієнтів та представлення ключ-значення.

Розглянемо перший підхід, нехай маємо *s*-слівне ціле числа *a*, тоді поліном буде мати вигляд:

$$A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_sx^s$$

Наприклад, представимо число 348 у вигляді поліному з коефіцієнтами: $A(x) = 8 + 4x + 3x^2$, де *x=10*

Інший підхід представлення поліному є ключ-значення. Маємо набір значень $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ причому виконуються наступні умови:

– для всіх $i \neq j, x_i \neq x_j$. Тобто, всі точки унікальні;

– для всіх k, $y_k = A(x_k)$.

Наприклад, маємо такі поліноми:

$$A(x) = x^3 - 2x + 1$$

$$B(x) = x^3 + x^2 + 1$$

візьмемо 7 коефіцієнтів $x_k = \{-3, -2, -1, 0, 1, 2, 3\}$.

Далі представимо поліноми *A* та *B* у вигляді ключ-значення, тобто для кожного x_k обчислимо значення поліному:

A: (-3, -20), (-2, -3), (-1, 2), (0, 1), (1, 0), (2, 5), (3, 22)

B: (-3, -17), (-2, -3), (-1, 3), (0, 1), (1, 3), (2, 13), (3, 37)

Якщо виконувати множення двох поліномів, що представленні як ключ-значення, то множення буде по точкове, тобто для кожного x_k маємо значення $A(x_k)$ та $B(x_k)$, то для того щоб отримати значення $C(x_k)$, достатньо виконати $C(x_k) = A(x_k) \cdot B(x_k)$ і час такого множення буде складати *O(n)*.

Повернемося до попереднього прикладу і виконаємо множення $C = A \cdot B$:

C: (-3, 340), (-2, 9), (-1, 2), (0, 1), (1, 0), (2, 65), (3, 814)

Приведення поліному до стандартного вигляду з коефіцієнтами називається інтерполяцією, пілся того як отримали поліном з коефіцієнтами можна легко привести поліном до цілого числа, що і є результатом множення двох цілих чисел.

Якщо представити *A* та *B,* як вектори, тоді вектор *C* буде називатися математичною згорткою *A* та *B* (представляється, як $A \otimes B$).

2.2.2 Постановка задачі

Маємо два поліноми:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n$$

потрібно обчислити $C(x) = A(x) \cdot B(x)$, за час менший $O(n^2)$, що є часом прямолінійного множення, як показано в прикладі:

$$\begin{array}{rrrrrrrrrrrrr} & & & & & & 2x^2 & + & 0x & + & 5 \\ & & & & & & 1x^2 & + & 2x & + & 3 \\ & & & & & & - & - & - & - & - & - \\ & & & & & & 6 & & 0 & & 15 \\ & & & & 4 & & 0 & & 10 & & \\ & & 2 & & 0 & & 5 & & & & \\ - & - & - & - & - & - & - & - & - & - & - & - & - \\ 2x^4 & + & 4x^3 & + & 11x^2 & + & 10x & + & 15 & & & & \end{array}$$

Більш загально, потрібно обчислити коефіцієнти c_i, коли

$$C(x) = A(x) \cdot B(x)$$

$$C(x) = c_0 + c_1 + c_2 x^2 + \ldots + c_{2n} x^{2n},$$

тоді $c_i = a_0 b_i + a_1 b_{i-1} + \ldots + a_i b_0$.

2.2.3 Перетворення Фур'є та теорема про згортку

Перетворення Фур'є – інтегральне перетворення однієї комплексної функції дійсної змінної на іншу, що дозволяє представити практично будь-яку функцію у вигляді таких тригонометричних функцій, як синус та косинус.

Теорема про згортку. Нехай маємо функції f та g і їх згортку $\{f \otimes g\}$, позначимо перетворення Фур'є оператором $'\zeta'$. Тоді $\zeta(f)$ та $\zeta(g)$ є перетворенням Фур'є функцій f та g. Відповідно до теореми про згортку маємо:

$$\zeta(f \otimes g) = \zeta(f) \cdot \zeta(g)$$

де $\zeta(f) \cdot \zeta(g)$ - є по точковим множенням, що було розглянуто раніше.

2.2.4 Алгоритм множення

Ідея даного підходу полягає в тому, що множення виконується не для заданих вхідних даних, а для іншого представлення поліному, для якого множення буде виконуватися за час $O(n)$. Тоді асиптотична складність алгоритму буде залежати саме від методів переходу від одного предсталеня до іншого, в нашому випадку від представлення з коефіцієнтами до представлення ключ-значення.

Опишмо даний алогоритм:

На вході: цілі числа *a* та *b.*

На виході: $\mathrm{c} = a \cdot b$

Крок 1. Представимо числа *a* та *b* у вигляді поліномів, як це показано у п.1.3.2.

Крок 2. Виконаємо оцінку коефіцієнтів.

Крок 3. Виконаємо по точкове множення.

Крок 4. Виконаємо інтерполяцію для коефіцієнтів поліному *C.*

Крок 5. Приведемо поліном до цілого числа.

Отримане число на кроці 5 буде результатом множення двох чисел *a* та *b.*

Загальна схема алгоритму показана на рисунку 2.1.

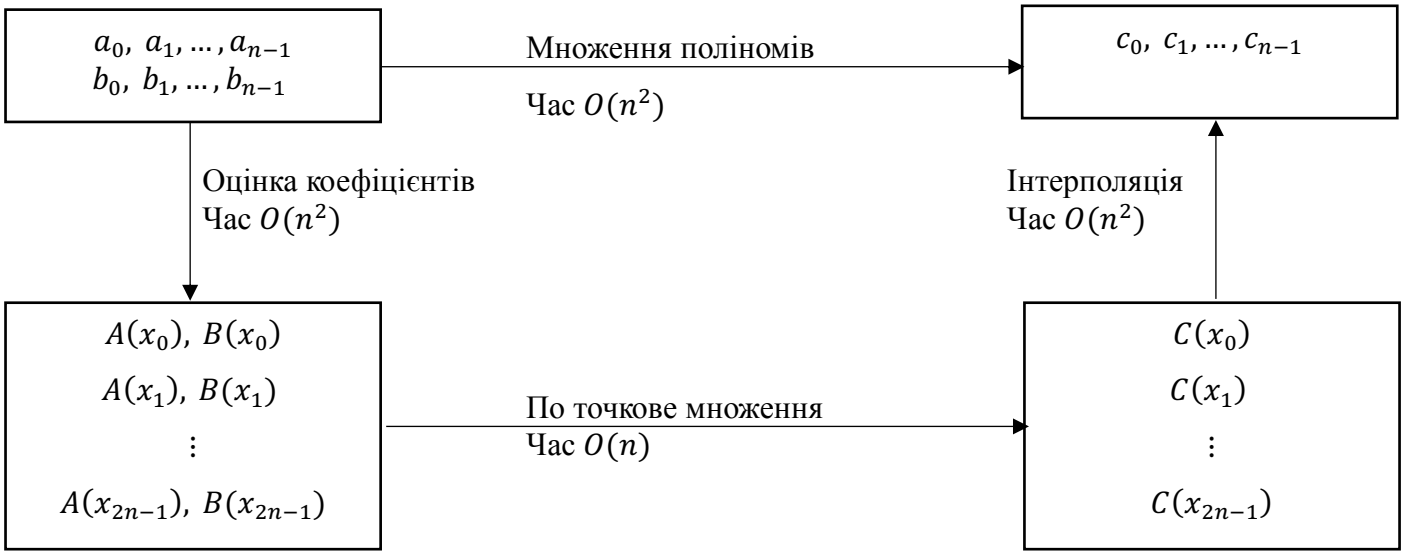

Рисунок 2.1 – Загальна схема алгоритму множення чисел

Як бачимо складність алгоритму залежить саме від переходу від одного представлення поліному до іншого, в нашому випадку від представлення з коефіцієнтами до ключ-значення на етапі оцінки коефіцієнтів та в зворотньому на етапі інтерполяції. Тому загальна ефективність алгоритму буде залежати від ефективності конвертації між двома представленнями.

Тому для переходу від одного представлення до іншого використаємо алгоритм швидкого перетворення Фур'є, як вже відомо складність $O\big(n \cdot log(n)\big)$.

Отже, загальний алгоритм буде мати вигляд:

Крок 1. Оцінюємо значення поліномів *A* та *B* в *2n* точках, поліноми *A* та *B* повинні бути заповненні нулями до ступеня рівного *2n*, використовуючи алгоритм швидкого перетворення Фур'є. Складність $O\big(n \cdot log(n)\big)$.

Крок 2. Знаходимо значення поліному *C* в *2n* точках, виконуючи по точкове множення *A* на *B*. Складність $O(n)$.

Крок 3. Для інтерполяції застосуємо обернене перетворення Фур'є, використавши швидкий алгоритм перетворення Фур'є. Складність $O\big(n \cdot log(n)\big)$.

Схема модифікованого алгоритму показана на рисунку 2.2.

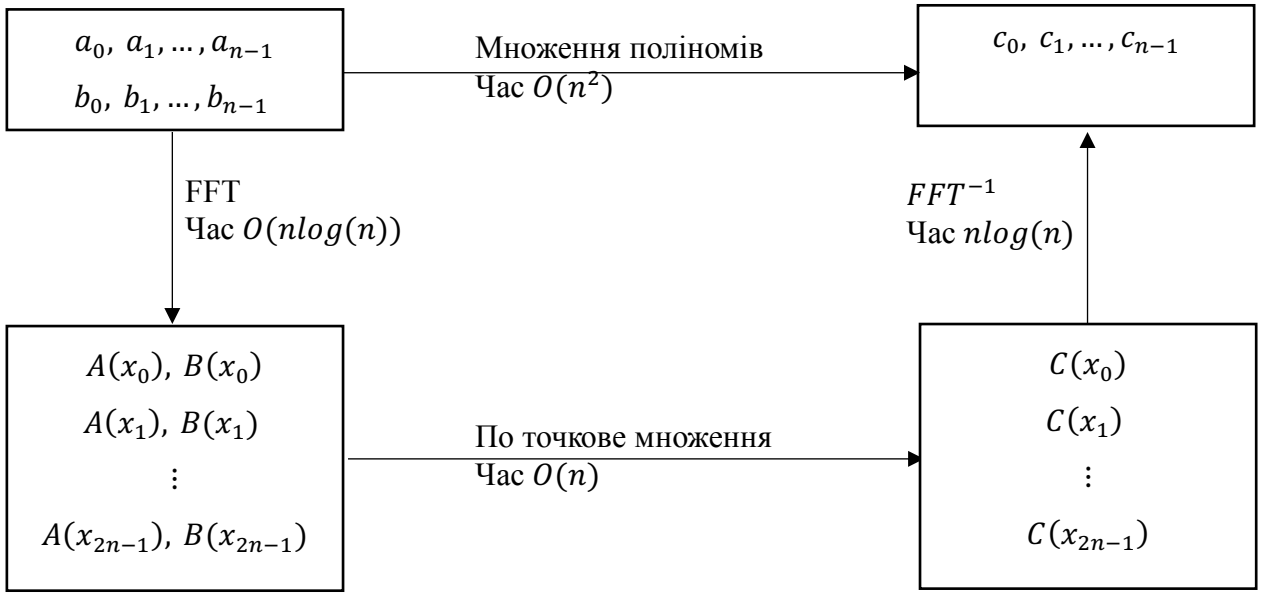

Рисунок 2.2 – Модифікована схема алгоритму

Використавши даний підхід, отримаємо прискорення по часу множення двох чисел від $O(n^2)$ до $O\big(n \cdot log(n)\big)$.

2.2.5 Проведення експериментів

Програмний продукт був реалізований у вигляді бібліотеки з двома методами множення чисел: перший – це стандартний метод множення чисел, другий – множення чисел, застосувавши швидке перетворення Фур'є.

Експеримент проводився на обладнанні з наступними характеристиками:

- процесор з тактовою частотою 2.2 ГГц;
- об'єм оперативної пам'яті 16 ГБ;
- операційна система macOS High Sierra.

Результати експерименту наведені в таблиці 2.2, де *n* – степінь двох чисел множення.

Таблиця 2.2 – Результати експериментів

n	*FFT* (мс)	Стандартний підхід (мс)
64	0.092	0.102
128	0.122	0.134

256	0.147	0.175
512	0.185	0.319
768	0.246	0.599
1024	0.281	0.924
1280	0.314	1.367
1536	0.373	1.810
1792	0.409	2.368
2048	0.427	3.018
4096	0.678	12.001
8192	1.568	45.863
16384	2.983	188.022

Результати експерименту також представленні у графічному вигляді на рисунку 2.3.

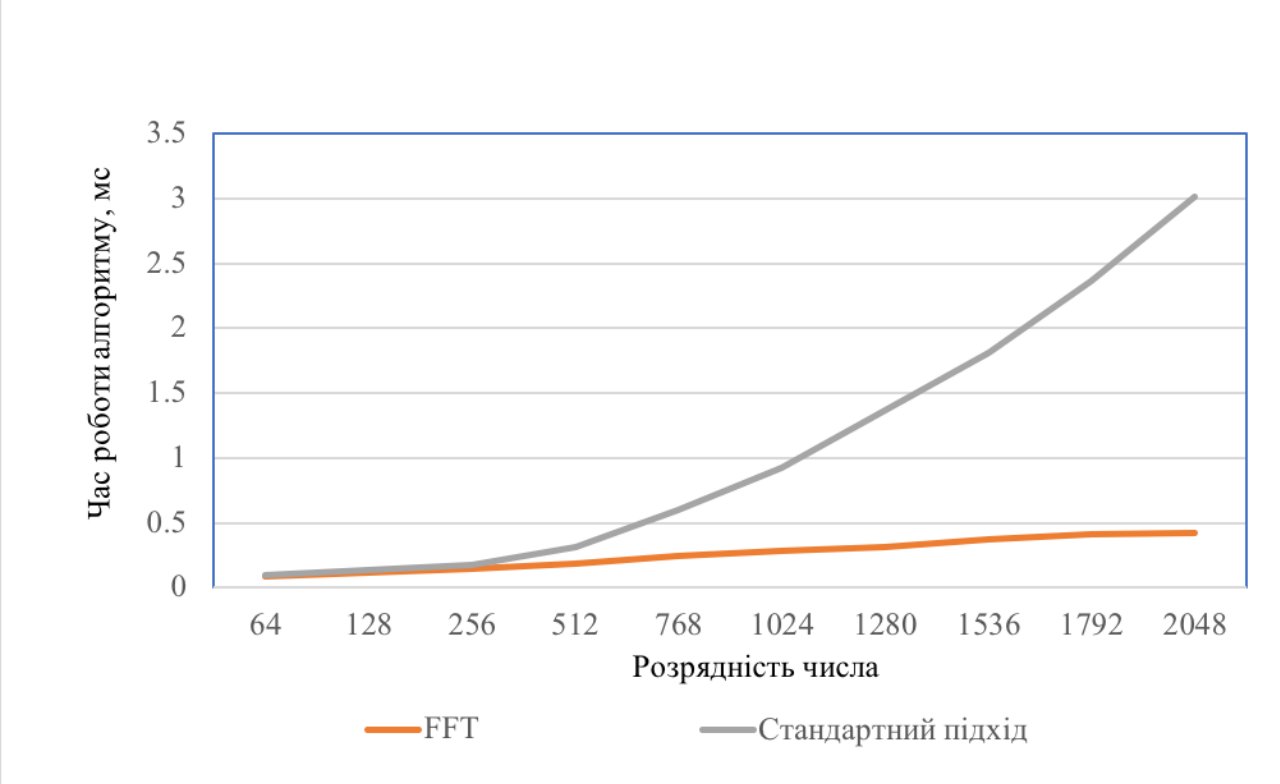

Рисунок 2.3 – Порівняння часу роботи двох алгоритмів множення

2.3 Висновки до розділу

В даному розділі описано два підходи до множення багаторозрядних чисел. Теоретично та експериментально було показано, що підхід множення чисел з використанням алгоритму швидкого перетворення Фур’є працює швидше стандартного підходу множення двох чисел, для багато розрядних чисел.

3 ОБЧИСЛЕННЯ ЕКСПОНЕНТИ ЗА МОДУЛЕМ

Перше правило обчислення експоненти за модулем:

$$C = x^n mod\ M$$

- не обчислювати x^n з наступним виконанням ділення з методом отримання лишку.

Будь-який проміжний результат на кожному кроці обчислення степеня треба наводити за модулем *M*. Це пов'язано великими вимогами до пам'яті для двійкового числа x^n. Якщо x^n мають 256 бітів кожний, нам знадобиться

$$log_2 x^n = n log_2 x \approx 2^{256} \cdot 256 = 2^{264} \approx 10^{80}$$

бітів, щоб зберегти x^n. Що є достатньо великим значенням для зберігання в оперативній пам'яті навіть для сучасних комп'ютерів.

Питання: скільки множень за модулем треба виконати, щоб обчислити $x^n\ mod\ M$? Найпростіший шлях обчислення $C = x^n mod\ M$ це почати з $C = x\ mod\ M$ і послідовно виконувати операції множення за модулем

$$C = C \cdot x\ mod\ M$$

до ти пір поки не отримаємо $C = x^n mod\ M$. Цей метод потребує *n - 1* множень за модулем, щоб обчислити $C = x^n mod\ M$, що вимагало б багато часу для великих *n.* Наприклад, якщо нам потрібно обчислити $C = x^n mod\ M$, цей метод обчислює всі степені *x* аж до 15:

$$x \to x^2 \to x^3 \to x^4 \to x^5 \to x^6 \to \ldots \to x^{15},$$

що потребує 14 множень. Однак, немає потреби обчислювати усі степені, щоб отримати x^{15}. Розглянемо більш швидкий метод обчислення x^{15}:

$$x \to x^2 \to x^3 \to x^5 \to x^7 \to x^{14} \to x^{15},$$

який потребує 6 множень. Цей метод працює не тільки для певних показників, його можна використовувати для обчислення x^n при довільних *n.* Метод називається *бінарним* або методом "підносьдо квадрату та множ".

3.1 Бінарні алгоритми

3.1.1 Алгоритм, який ґрунтується на зчитуванні "зліва направо"

У цьому пункті обговоримо задачу ефективного обчислень x^n за заданими *x* та *n*, де *n* - додатне ціле число.

Запишемо *n* у двійковій системі числення і замінимо кожну цифру "1" парою букв *SX,* а кожну цифру "0" - буквою *S,* після чого викреслимо крайню ліву пару букв *SX*. Результат перетворюється в правило обчислення x^n, якщо букву "*S"* трактувати як операцію піднесення до квадрату, а букву *"X"* – як операцію множення на x. Наприклад, якщо *n* = 23, то йог двійковим представленням буде 10111; будуємо послідовність *SX S SX SX SX*, видаляємо з неї початкову пару *SX* і в результат отримуємо таке правило обчислення: *S SX SX SX*. Згідно цього правила, ми повинні "піднести *x* до квадрату, потім знову піднести до квадрату, потім помножити на *x*, піднести до квадрату, помножити на *x,* піднести до квадрату і, решті, помножити на *x* при цьому послідовно обчислюємо $x^2, x^4, x^5, x^{10}, x^{11}, x^{22}, x^{23}$.

Для довільного *k*-бітного числа *n* при $n_{k-1} = 1$,

$$\left(n = (n_{k-1}n_{k-2} \dots n_1 n_0) = \sum_{i=0}^{k-1} n_i 2^i; n_i \in \{0,1\}\right)$$

бінарний метод потребує:

– Піднесень до квадрату: $k - 1$;

– Множень: *H(n)* – 1, де *H(n)* – *Hamming* – вага числа (кількість одиниць у двійковому розкладі).

Нехай $n > 0$, тоді $0 \leq H(n) - 1 \leq k - 1$. У припущенні, що $n_{k-1} = 1$, загальне число множень оцінюється таким чином:

– Максимальне: $(k - 1) + (k - 1) = 2(k - 1)$;

– Мінімальне: $(k - 1) + 0 = (k - 1)$;

– Середнє: $(k - 1) + 1/2(k - 1) = 3/2(k - 1)$.

Бінарний метод "*S* та *X*" обчислення x^n не вимагає ніякої додаткової оперативної пам'яті, за винятком пам'яті для збереження x і поточного проміжного результату.

3.1.2 Проведення експериментів

Програмний продукт був реалізований у вигляді бібліотеки з методом піднесення до степення за модулем.

Експеримент проводився на обладнанні з наступними характеристиками:

- процесор з тактовою частотою 2.2 ГГц;
- об'єм оперативної пам'яті 16 ГБ;
- операційна система macOS High Sierra.

Результати експерименту наведені в таблиці 3.1.

Таблиця 3.1. Результати експериментів

	Алгоритм "зліва направо"
k	T, мс
128	2282
256	4593
512	9691
768	15978
1024	18366
2048	21047

3.2 m-арні алгоритми обчислення за модулем

У бінарному алгоритмі показник *n* сканується послідовно по одному біту. Якщо сканувати по 2 біти відразу, отримаємо кватеріарний алгоритм, якщо по три, то октальний алгоритм і т.д.

У загальному випадку випадку, якщо сканується відразу $log_2 m$ бітів, то такий алгоритм називають *m*-арним.

m-арний алгоритм базується на *m*-арному розкладі показника, відповідно до якого цифри *n* скануються, а потім виконуються піднесення до квадрату (або до потрібного степення) та необхідні добутки. Коли *m* є степенем 2, реалізувати *m*-арний алгоритм досить просто, так як x^n обчислюється групуванням бітів у двійковому розкладі показника *n*. Нехай $n = (n_{k-1}n_{k-2} \dots n_1 n_0)$ двійковий розклад показника, який складається з *s* блоків по *r* бітів, коли $s \cdot r = k$. Якщо *r* не ділить *k*, то показник доповнюється нулями, найбільше *r-1* нулями. Тепер визначимо

$$F_i = (n_{ir+r-1}n_{ir+r-2} \dots n_{ir}) = \sum_{j=0}^{r-1} n_{ir+j} \cdot 2^j.$$

Зауважимо, що $0 \le F_i \le m-1$ і $n = \sum_{i=0}^{s-1} F_i \cdot 2^{ir}$. *m*-арний метод спочатку обчислює $x^{\omega} mod\ M$ для $\omega = 2, 3, \dots, m-1$. Потім двійкові розряди *n* скануються по *r* розрядів відразу, починаючи від старших значущих розрядів до молодших. На кожному кроці попередній результат підноситься до степеня 2^r і множиться на x^{F_i} за модулем *M*, де F_i— ненульове значення бітсекції.

Опишемо покроково *m*-арний алгоритм:

На вході: *x*, *n*, *M*

На виході: $c = x^n\ mod\ M$

Крок 1. Обчислити і зберегти $x^{\omega} mod\ M$ для всіх $\omega = 2, 3, \dots, m-1$;

Крок 2. Розкласти *n* в *s* *r*-бітових слова F_i для $i = 0, 1, 2, \dots, s-1$;

Крок 3. Обчислити $c = x^{F_{s-1}}\ mod\ M$;

Крок 4. Для всіх $i = s-2 \to 0$;

Крок 4a. Обчислити $\mathrm{c} = \mathrm{c}^{2^r}\ mod\ M$;

Крок 4b. Якщо $F_i \neq 0$, то $c = c \cdot x^{F_i}\ mod\ M$;

Крок 5. Результат *c*.

При *m = 2* цей метод перетворюється в бінарний метод “зліва-направо”.

Аналіз “бінарного” і “*m*-арного” алгоритмів наведено для випадку, коли двійкове зображення показників розбивається на групи по *r* розрядів ($m = 2^r$). Результат аналізу подано в порівняльній таблиці 3.2, яка дає змогу визначити

ефективне співвідношення між кількістю двійкових розрядів k та r. Порівняння двох вказаних методів проведене за кількістю операцій множення і піднесення до квадрату багаторозрядних чисел, які потрібні для обчислення експоненти.

Таблиця 3.2 – Порівняльний аналіз бінарного та m-арного методів

Розмірність показника	m-арний метод			Бінарний метод	
В бітах (k)	r	К-сть множень	К-сть піднесень до квадрату	К-сть множень	К-сть піднесень до квадрату
12	2	6	12	11	11
51	3	19	53	50	50
180	4	51	186	179	179
560	5	126	574	559	599
1600	6	297	1630	1599	1599

Крім програми виконання операції множення, корисно мати спеціальні програми для піднесення числа до квадрату, оскільки операція часто використовується і має додатковий резерв оптимізації часу її виконання.

Наведемо оцінки необхідних часових затрат на піднесення до n-слівного степеня за n-слівним модулем n-слівного числа бінарним і m-арним алгоритмами і проаналізуємо, який виграш може бути отриманий за рахунок оптимізації ключових операцій (множення і піднесення до квадрату за модулем).

Не важко показати, що у випадку 16-бітових слів потрібно виконати:

– за бінарним алгоритмом – $16n$ піднесень до квадрату і в середньому $8n$ множень за модулем;

– за m-арним алгоритмом (при довжині блока в p бітів) – $16n + 2^{p-1} - p - 1$ піднесень до квадрату і $16n/p + 2^{p-1} - 2$.

3.2.1 Кватернарний алгоритм

Так як розряди показника *n* скануються по два за один раз, то можна отримувати такі значення (00) = 0, (01) = 1, (10) = 2, (11) = 3. На кроці добутку (крок 4b) можуть вимагатись значення x^0, x^1, x^2, x^3. Таким чином, треба виконати деякі передобчислення, Щоб отримати x^2, x^3. Для прикладу, нехай *n* = 250 і поділ *n* на групи по два розряди має вигляд *n* = 250 = 11 11 10 10.

Таким чином, ми маємо *s*=4 (кількість груп $s = \frac{k}{r} = \frac{8}{2} = 4$). Зробимо передобчислення, як це показано в таблиці 3.3.

Таблиця 3.3 – Передобчислення кватернарного алгоритму

Розряди	ω	x^{ω}
00	0	1
01	1	x
10	2	$x \cdot x = x^2$
11	4	$x^2 \cdot x = x^3$

Кватернарний алгоритм ініціалізується наступним чином $c = x^{F_3} = x^3 \ mod \ M$ і обчислення $x^{250} \ mod \ M$ виконується як це показано в таблиці 3.4.

Таблиця 3.4 – Хід обчислення кватернарного алгоритму

i	F_i	Крок 4a	Крок 4b
2	11	$(x^3)^4 = x^{12}$	$x^{12} \cdot x^3 = x^{15}$
1	10	$(x^{15})^4 = x^{60}$	$x^{60} \cdot x^2 = x^{62}$
0	10	$(x^{62})^4 = x^{248}$	$x^{248} \cdot x^2 = x^{250}$

Кількість множень, потрібних в цьому методі для обчислення $x^{250} \ mod \ M$, дорівнює 2 + 6 + 3=11.

Розглянемо приклад обчислення $4^{250} \ mod \ 23$, як було зазначено раніше параметри даного алгоритму *m*=4, *s*=4, *k*=8, *r*=2, *n*=250, *x*=4, *M*=23.

Представимо число 250 у бінарному вигляді $(n_{10} = (n_{k-1}n_{k-2} \dots n_1 n_0)_2)$ $250_{10} = (1_7 1_6 \ 1_5 1_4 \ 1_3 0_2 \ 1_1 0_0)_2$.

Виконаємо передобчислення як було показано в таблиці 3.3 для $x = 4$. Результати передобчислень показані в таблиці 3.5.

Таблиця 3.5 – Передобчислення кватернарного алгоритму для $x = 4$

Розряди	ω	4^{ω}
00	0	1
01	1	x=4
10	2	$x^2 = 16$
11	4	$x^3 = 64$

Визначимо F_i за формулою $F_i = \sum_{j=0}^{r-1} n_{ir+j} \cdot 2^j$, як це показано в таблиці 3.6.

Таблиця 3.6 – Обчислення F_i

i	F_i
0	$\sum_{j=0}^{1} n_{0+j} \cdot 2^j = n_0 + 2n_1 = 0 + 2 = 2$
1	$\sum_{j=0}^{1} n_{2+j} \cdot 2^j = n_2 + 2n_3 = 0 + 2 = 2$
2	$\sum_{j=0}^{1} n_{4+j} \cdot 2^j = n_4 + 2n_5 = 1 + 2 = 3$
3	$\sum_{j=0}^{1} n_{6+j} \cdot 2^j = n_6 + 2n_7 = 1 + 2 = 3$

Виконуємо ініціалізацію $c = x^{F_3} = x^3 \; mod \; M = 4^3 mod \; 23 = 18$, далі виконуємо обчислення для $\mathrm{i} = \overline{\mathrm{s} - 2, 0}$, як це показано в таблиці 3.7.

Таблиця 3.7 – Хід обчислення кватернарного алгоритму

i	F_i	Крок 4a	Крок 4b
2	3	$c = c^{2^r} \ mod \ M = (18)^4 mod \ 23 = 4$	$c = c \cdot x^{F_i} \ mod \ M = = 4 \cdot 4^3 mod \ 23 = 3$
1	2	$c = c^{2^r} \ mod \ M = (3)^4 mod \ 23 = 12$	$c = c \cdot x^{F_i} \ mod \ M = = 12 \cdot 4^2 mod \ 23 = 8$
0	2	$c = c^{2^r} \ mod \ M = (8)^4 mod \ 23 = 2$	$c = c \cdot x^{F_i} \ mod \ M = = 2 \cdot 4^2 mod \ 23 = 9$

Отже, маємо $4^{250} \ mod \ 23 = 9$ за 11 множень.

3.2.2 Октальний алгоритм

У цьому алгоритмі двійкові представлення поділяються на групи по три розряди. Наприклад, для показника *n* = 250 такий поділ має вигляд: n = 250 = 011 111 010 після додавання зліва одного нуля. Це дає $s = \frac{k}{r} = \frac{9}{3} = 3$. На кроці передобчислення обчислюємо $x^{\omega} \ mod \ M$ для всіх $\omega = 2, 3, 4, 5, 6, 7$. Результати представленні в таблиці 3.8.

Таблиця 3.8 – Передобчислення октального алгоритму

Розряди	ω	x^{ω}
000	0	1
001	1	x
010	2	$x \cdot x = x^2$
011	3	$x^2 \cdot x = x^3$
100	4	$x^3 \cdot x = x^4$
101	5	$x^4 \cdot x = x^5$
110	6	$x^5 \cdot x = x^6$
111	7	$x^6 \cdot x = x^7$

Октальний алгоритм ініціалізується наступним чином: с $= x^{F_2} = x^3 \; mod \; M$, а процес обчислення x^{250} показано в таблиці 3.9.

Таблиця 3.9 – Хід обчислення кватернарного алгоритму

i	F_i	Крок 4a	Крок 4b
1	111	$(x^3)^8 = x^{24}$	$x^{24} \cdot x^7 = x^{31}$
0	010	$(x^{31})^8 = x^{248}$	$x^{248} \cdot x^2 = x^{250}$

Обчислення $x^{250} \; mod \; M$ октальним алгоритмом потребує 6 + 6 + 2 = 14 добутків за модулем. Однак зауважимо, що ми використали не усі степені $x^{\omega} \; mod \; M$ для $\omega = \overline{2,7}$, які були обчислені попередньо. Отже, можна модифікувати крок 1 m-арного алгоритму і обчислити попередньо $x^{\omega} \; mod \; M$ тільки для таких ω, які з'являються в розкладі двійкового представлення показника на бітсекції. Наприклад, для n = 250 це (001) = 3, (111) = 7, (010) = 2. Степені для них ми можемо знайти за допомогою лише 4-х добутків, як це показано в таблиці 3.10.

Таблиця 3.10 – Передобчислення октального алгоритму для 4 добутків

Розряди	ω	x^{ω}
000	0	1
001	1	x
010	2	$x \cdot x = x^2$
011	3	$x^2 \cdot x = x^3$
100	4	$x^3 \cdot x = x^4$
111	7	$x^4 \cdot x^3 = x^7$

Це дає загальну кількість добутків, які потрібні октальному методу для обчислення $x^{250} \; mod \; M$, рівну 4 + 6 + 2 = 12. Метод обчислення $x^n \; mod \; M$ з попереднім обчисленням $x^{\omega} \; mod \; M$ тільки для тих ω, які з'являються в поділі показника на бітсекції зветься адаптивним (або залежним від вихідних даних) алгоритмом.

3.2.3 Проведення експериментів

Програмний продукт був реалізований у вигляді бібліотеки з методами піднесення до степення за модулем “зліва направо” та m-арний.

Експеримент проводився на обладнанні з наступними характеристиками:

- процесор з тактовою частотою 2.2 ГГц;
- об’єм оперативної пам’яті 16 ГБ;
- операційна система macOS High Sierra.

Результати експерименту наведені в таблиці 3.11.

Таблиця 3.11 – Результати експериментів

ç	Алгоритм “зліва направо”	m-арний алгоритм	
k	*T*, мс	*r*	*T*, мс
128	2282	4	1876
256	4593	4	3816
512	9691	5	7607
768	15978	5	13158
1024	18366	5	14538
2048	21047	6	16209

Результати експерименту також наведені на рисунку 3.1.

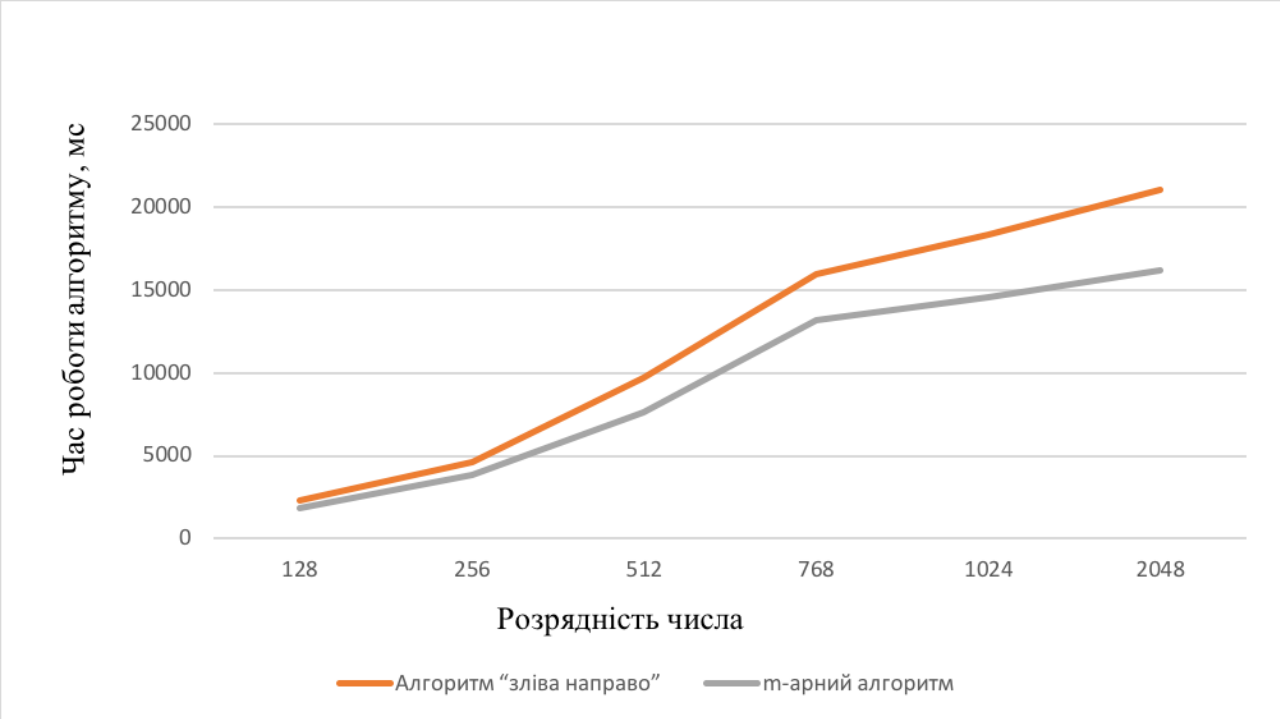

Рисунок 3.1 – Результати експериментів для алгоритмів зліва-направ та m-арного алгоритму

Результатами експериметів було підтверджено, що m-арний алгоритм виконує піднесення степення за модулем швидше методу зліва-направо, завдяки тому що кількість множень суттєво скорочується.

3.3 Адаптивні m-арні методи

Адаптивні методи - це методи, які будують процес обчислення в залежності від вхідної інформації. У випадку піднесення до степеня адаптивні методи будуть змінювати свою структуру залежно від значення показника *n*. Кількість множень на етапі передобчислень може бути зменшена. Існують також адаптивні алгоритми, в яких розбитні на нульові і ненульові слова робиться так, щоб зменшити кількість множень в *m*-арному методі.

3.3.1 Зменшення числа множень на кроці передобчислень

Як тільки отримано двійкове представлення показника, розіб'ємо його на групи по r бітів кожна. Потім зробимо передобчислення і отримаємо $x^{d_j} mod\ M$ тільки для тих d_j, які з'являються у двійковому розкладі. Розглянемо такий показник 1011 0011 0111 1000 для k=16 та r=4. Нам потрібно обчислити

$x^{\omega}\ mod\ M$ тільки для $\omega = 3, 7, 8, 11$, що могло б послідовно бути отримано таким чином:

$$x^2 = x \cdot x$$
$$x^3 = x^2 \cdot x$$
$$x^4 = x^2 \cdot x^2$$
$$x^7 = x^4 \cdot x^3$$
$$x^8 = x^4 \cdot x^4$$
$$x^{11} = x^8 \cdot x^3$$

На це треба 6 множень. m-арний метод, який обчислює степені для всіх значень ω на кроці передобчислень потребує 16 – 2 = 14 множень. Кількість множень, які можна зекономити в m-арному методі, обмежена зверху величиною $m - 2 = 2^r - 2$, що має місце, коли всі біт-секції мають вигляд: 0001 0001 0001 0001.

Не означає, що ми нічого не передобчислюємо, а використовуємо тільки x. Це відбувається досить рідко. Взагалі, ми повинні обчислити $x^{\omega}\ mod\ M$ для всіх $\omega = \omega_0, \omega_1, \dots, \omega_{m-1}$. Якщо різні елементи множини $\{\omega | i = 0, 1, \dots, m-1\}$ приймають усі можливі значення $2, 3, \dots, 2^r - 1$, то економія неможлива. Ми виконуємо $2^r - 2$ множень і отримуємо усі значення.

3.3.2 Алгоритм змінних вікон

Алгоритми обчислення експоненти з змінними вікнами спочатку здійснюють декомпозицію показника n на нульові та ненульові слова (вікна) F_i довжини $L(F_i)$. Кількість вікон m не може дорівнювати k/r. У загальному випадку також не потрібно щоб вікна були однакової довжини. Беремо r рівним довжині самого довгого вікна, тобто $d = \max\big(L(F_i)\big), i = 0,1, \dots, k-1$. Однак, якщо F_i довільне ненульове вікно, то самий молодший значущий біт F_i, повинен бути рівним 1, тому, що ми розбиваємо показники, починаючи з самого молодшого значущого біта. Таким чином кількість множень при переодобчисленні (крок 1) скоротиться приблизно на половину, так як степені x^{ω} обчислюються тільки для непарних показників ω.

Покроковий опис алгоритму змінних вікон:

На вході: *x*, *n*, *M*

На виході: $c = x^n \ mod \ M$

Крок 1. Обчислити і зберегти $x^{\omega} mod \ M$ для всіх $\omega = 3, 5, 7, \dots, 2^r - 1$;

Крок 2. Розкласти *n* на нульові та ненульові вікна F_i довжини $L(F_i)$ для $i = 0, 1, 2, \dots, m - 1$;

Крок 3. Обчислити $c = x^{F_{m-1}} \ mod \ M$;

Крок 4. Для всіх $i = m - 2 \rightarrow 0$;

Крок 4a. Обчислити $\mathrm{c} = \mathrm{c}^{2^{L(F_i)}} \ mod \ M$;

Крок 4b. Якщо $F_i \neq 0$, то $c = c \cdot x^{F_i} \ mod \ M$;

Крок 5. Результат *c*.

Існують дві стратегії. Відмінність їх полягає в наступному: чи повинна довжина ненульових вікон бути однаковою (= r), або вікна можуть мати різну довжину ($\leq r$). У наступних розділах ми дамо алгоритмічний опис цих двох стратегій розбиття.

3.3.3 Ненульові вікна однакової довжини (CLNW)

Алгоритм розбиття запропонований Д.Кнутом. Він сканує біти показника від молодшого до старшого. У будь-який момент алгоритм формує або нульове вікно *(ZW),* або ненульове (NW).Алгоритм описаний нижче:

ZW: Перевіряємо вхідний одиничний біт: якщо він дорівнює 0, то лишаємось в ZFF, якщо ні, то йдемо в *NW*.

NW: Лишаємось в *NW* до того часу, поки не наберемо *r* бітів. Потім перевіряємо один вихідний біт: якщо він дорівнює 0, то йдемо в ZW; інакше – в NW.

Зауважимо, що в кроці *NW* треба відрізняти знаходження на кроці *NW* та вихід з нього. Перше означає, що ми продовжуємо формувати те ж саме ненульове вікно, а в другому випадку це означає початок нового ненульового вікна. Алгоритм *CLNW* генерує нульові вікна довільної довжини, а ненульові - довжини *r*.

Не може бути двох суміжних нульових вікон; вони обов'язково зчеплюються, а ненульові - можуть бути суміжними. Наприклад, для r = 3 розбиваємо *n* = 3665 = (111001010001) наступним чином: n = 111 00 101 0 001.

Спочатку алгоритм *CLNW* виконує множення при передобчисленні і ми отримуємо $x^{\omega}\ mod\ M$ тільки для $\omega = 3, 5, 7$, як це показано в таблиці 3.12.

Таблиця 3.12 – Передобчислення алгоритму *CLNW*

Розряди	ω	x^{ω}
001	1	x
010	2	$x \cdot x = x^2$
011	3	$x^2 \cdot x = x^3$
101	5	$x^3 \cdot x^2 = x^5$
111	7	$x^5 \cdot x^2 = x^7$

Алгоритм ініціалізується $c = x^{F_{m-1}}\ mod\ M = x^7 mod\ M$ і потім виконується для обчислення $x^{3565}\ mod\ M$ кроки, що показані в таблиці 3.13.

Таблиця 3.13 – Хід обчислення алгоритму *CLNW*

i	F_i	$L(F_i)$	Крок 4a	Крок 4b
3	00	2	$(x^7)^4 = x^{28}$	x^{28}
2	101	3	$(x^{28})^8 = x^{224}$	$x^{224} \cdot x^5 = x^{229}$
1	0	1	$(x^{229})^2 = x^{458}$	x^{458}
0	001	3	$(x^{258})^8 = x^{3664}$	$x^{3664} \cdot x = x^{3665}$

Таким чином, виконується всього 4 + 9 +2 = 15 множень за модулем.

В таблиці 3.14 показано середнє число множень для m-арного алгоритму та CLNW. Стовпчик для m-арного методу містить оптимальні значення r для кожного *k*. Природно чекати, що для алгоритму *CLNW* також існують оптимальні значення *r* для кожного *k.* Вони також включені в таблицю. Останній стовпчик таблиці містить відсоток відмінності в середній кількості множень. Стратегія *CLNW* розбиття показника зменшує середнє число множень на 3.7 % для

$128 \leq k \leq 2048$. Без урахування часу на розбиття показника, число біт-операцій, потрібних для розбиття, пропорційне k.

Таблиця 3.14 – Порівняння числа множень для m-арного алгоритму та CLNW

	m-арний		CLNW		$\frac{T - T_1}{T}$
k	r	T	r	T_1	%
128	4	168	4	156	7.14
256	4	326	5	308	5.52
512	5	636	5	607	4.56
768	5	941	6	903	4.04
1024	5	1247	6	1195	4.17
1280	6	1546	6	1488	3.75
1536	6	1844	6	1780	3.47
1792	6	2142	7	2072	3.27
2048	6	2440	7	2360	3.28

3.3.4 Ненульові вікна різної довжини (VLNW)

Стратегія розбиття *CLNW* починає визначати ненульове вікно, коли зустрічається 1. Хоча б і всі *r* - 1 вхідних бітів можуть бути рівні нулю, алгоритм продовжує приєднувати їх до поточного ненульового вікна. Наприклад, *r*=3, показник *n*=(1110010100001) розглядається, як *n* = 111 00 101 0 001.

Однак якщо ми приписуємо ненульові вікна різної довжини, то можемо розбити це число як *n* = 111 00 101 000 1.

Покажемо, що ця стратегія також зменшує середнє число ненульових вікон. Стратегія *VLNW* запропонована J. Bos і M.Coster. Вона потребує, щоб під час формування ненульового вікна *(NW)* ми переключались в *ZW*, коли всі біти, що залишилися - нульові. Стратегія *VLNW* має два цілих параметри:

r - максимальна довжина ненульового вікна;

q - мінімальна кількість нулів, потрібних для переключення в *ZW*.

Алгоритм має вигляд:

ZW: Перевіряємо один біт на вході: якщо він нульовий, то залишаємось в *ZW,* в іншому випадку йдемо на *NW*.

NW: Перевіряємо q бітів на вході: якщо всі вони нулі, то йдемо в ZW, інакше лишаємося в NW. Нехай $r = l \cdot q + d + 1$, де $l < d \leq q$. Лишаємося в NW, поки не досягнемо $l \cdot q + 1$ бітів. На останньому кроці кількість вхідних бітів дорівнює d. Якщо вони всі нульові, йдемо в ZW; інакше лишаємося в NW. Після того, як всі r бітів зібрані, перевіряємо один біт на вході: якщо він дорівнює нулю, йдемо в ZW; в протилежному випадку йдемо в NW.

Процедура розбиття *VLNW* формує ненульові вікна, які починаються з 1 і закінчуються 1. Два ненульових вікна можуть бути сусідами; однак ненульове вікно буде обов'язково мати r бітів. Наприклад, нехай r = 5, q = 2, тоді $5 = 1 + 1 \cdot 2 + 2$, отже, l=1, а d=2. Відповідно до цих параметрів розбиття буде виглядати наступним чином:

101 0 11101 00 101 10111 000 000 1 00 111 000 1011.

Тепер нехай r = 10, q = 4, тоді l = 2, d = 1, і ми маємо

1011011 0000 11 0000 11110111 00 1111110101 0000 11011.

Для підрахунку середнього числа множень, процес *VLNW* може моделюватись за допомогою ланцюгів Маркова, подібно процесу *CLNW*. Цей аналіз був зроблений в [24], і було відраховано середнє число множень для $128 \leq k \leq 2048$. В таблиці 3.15 наведено ці значення разом з оптимальними значеннями r і q, а також, порівняння їх з середнім числом множень в m-арному методі при оптимальному r. Експерименти показали, що найкращі значення q знаходяться між 1 та 3 для $128 \leq k \leq 2048$ і $4 \leq r \leq 8$. Алгоритм *VLNW* потребує множень на 5.8 % менше, ніж m-арний матод.

Таблиця 3.15 – Порівняння числа множень для m-арного алгоритму та VLNW

	m-арний		VLNW, T_2/k				$\frac{T_2-T}{T_2}$ для q^*
k	r	T/k	r	$q=1$	$q=2$	$q=3$	%
128	4	1.305	4	1.204	1.203	1.228	7.82
256	4	1.270	4	1.184	1.185	1.212	6.77
512	5	1.241	5	1.163	1.175	1.162	6.37
768	5	1.225	5	1.155	1.167	1.154	5.80
1024	5	1.217	6	1.148	1.1146	1.157	5.83
1280	6	1.207	6	1.142	1.140	1.152	5.55
1536	6	1.200	6	1.138	1.136	1.148	5.33
1792	6	1.195	6	1.136	1.134	1.146	5.10
2048	6	1.191	6	1.134	1.132	1.144	4.95

Алгоритми з змінними вікнами легко програмувати, включаючи незначні додаткові обчислення. Зменшення числа множень при цьому значне, наприклад, для n = 612, m-арний алгоритм потребує 636 множень, тоді як алгоритм *CLNW* і *VLNW* відповідно 607 і 595 множень.

3.3.5 Метод дерева степенів

Метод запропонований Д. Кнутом [5]. Відповідний алгоритм будує дерево степенів відповідно деякій евристиці. Вузли дерева помічаються додатними цілими числами, починаючи з 1. Кореневі дерева привласнюється значення рівне 1. Припустимо, що дерево побудоване до k-го рівня. Розглянемо вузли n k-го рівня зліва направо. Будуємо $(k+1)$-й рівень, приєднуючи до вузла n вузли $n + a_1, n + a_2, \dots, n + a_k$, де $a_1, a_2, \dots, a_k$ – шлях від кореня дерева до n (зауважимо, що $a_1 = 1$ і $a_k = l$), виключаючи повторення вузлів, які вже з'явились на дереві. Дерево степенів, яке містить 5 рівнів представлене на рисунку 3.2.

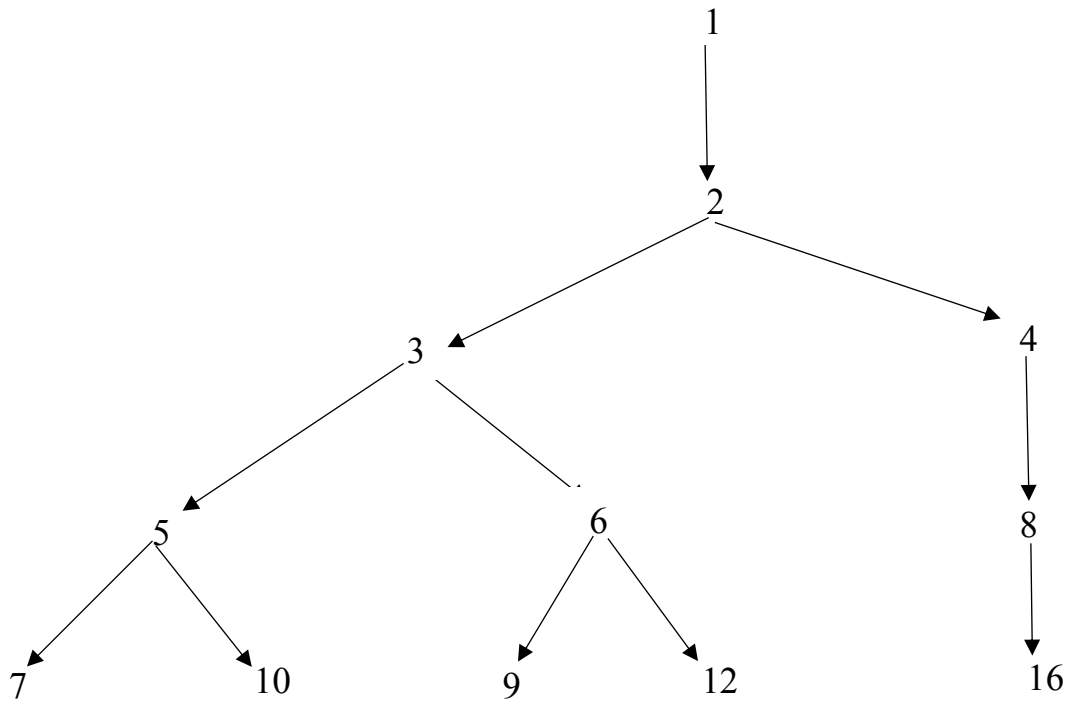

Рисунок 3.2 – Дерево степенів

Для того, щоб обчислити x^n, розміщуємо *n* на дереві степенів. Послідовність показників, які потрібні для обчислення x^n, знаходиться на шляху від кореня до *n*. Наприклад, обчислення x^{18} потребує 5 множень:

$$x \longrightarrow x^2 \longrightarrow x^3 \longrightarrow x^6 \longrightarrow x^9 \longrightarrow x^{18}.$$

Для певних значень *n* метод дерева степенів потребі меншого числа множень. Так, для обчислення x^{23} методом деревом степенів потрібно 6 множень:

$$x \longrightarrow x^2 \longrightarrow x^3 \longrightarrow x^5 \longrightarrow x^{10} \longrightarrow x^{13} \longrightarrow x^{23}.$$

В той же час, оскільки 23 = (10111), бінарному методу знадобиться 4 + 3 = 7 множень:

$$x \longrightarrow x^2 \longrightarrow x^4 \longrightarrow x^8 \longrightarrow x^{16} \longrightarrow x^{20} \longrightarrow x^{22} \longrightarrow x^{23}.$$

3.3.6 Проведення експериментів

Програмний продукт був реалізований у вигляді бібліотеки з методами піднесення до степення за модулем “зліва направо” та m-арний.

Експеримент проводився на обладнанні з наступними характеристиками:

- процесор з тактовою частотою 2.2 ГГц;
- об’єм оперативної пам’яті 16 ГБ;

– операційна система macOS High Sierra.

Результати експерименту наведені в таблиці 3.16.

Таблиця 3.16 – Результати експериментів

	Алгоритм “зліва направо”	m-арний алгоритм		CLNW	
k	T, мс	r	T, мс	r	T_1, мс
128	2282	4	1876	4	1782
256	4593	4	3816	5	3631
512	9691	5	7607	5	7456
768	15978	5	13158	6	12124
1024	18366	5	14538	6	13849
2048	21047	6	16209	7	15210

Результати експерименту також наведені на рисунку 3.3.

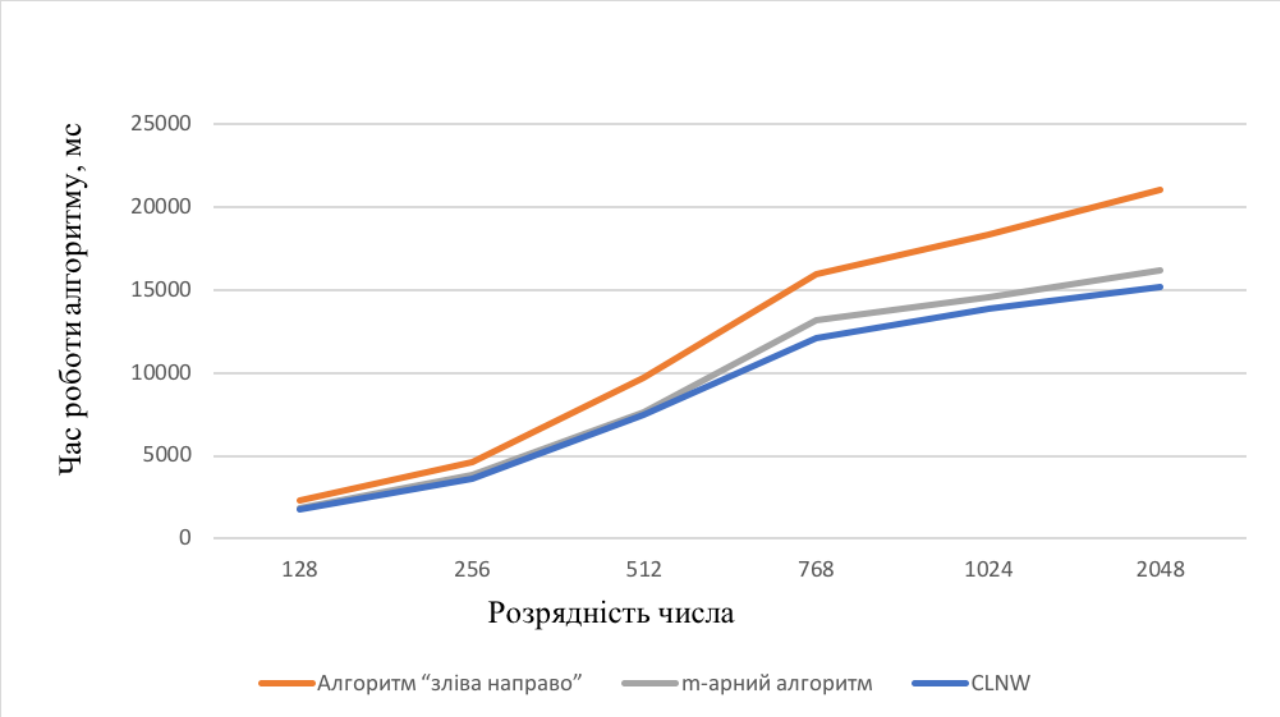

Рисунок 3.3 – Результати експериментів для алгоритмів зліва-направ та m-арного алгоритму

Результатами експериметів було підтверджено, що адаптивний алгоритм, а саме CLNW виконує піднесення степення за модулем швидше методу зліва-направо, завдяки тому що кількість множень суттєво скорочується.

3.4 Висновки до розділу

В даному розділі описано підходи до піднесення до степення цілого числа за модулем. Теоретично та експериментально було показано, що найкращими за швидкодією є m-арні алгоритмі та адаптивні m-арні алгоритмі, час роботи суттєво зменшується для багаторозрядних чисел.

4 ЗАСТОСУВАННЯ ЕФЕКТИВНИХ ЗА СКЛАДНІСТЮ АЛГОРИТМІВ ВИКОНАННЯ ОПЕРАЦІЙ НАД БАГАТОРОЗРЯДНИМИ ЧИСЛАМИ

Викладені вище ефективні за складністю алгоритме виконання операцій над багаторозрядними числами потрібні для підвищення продуктивності криптосистем з відкритими ключами, алгоритмів електронного цифрового підпису, криптографічних протоколів, здійснення високоточних обчислень та розв'язанні інших задач захисту інформації, обчислювальної, прикладної та дискретної математики.

4.1 Відкрите розповсюдження ключів

Як одну з можливих односторонніх функцій Діффі і Хеллман запропонували функцію дискретного піднесення до степеня

$$f(x) = a^x \ mod \ p$$

де *x* — ціле число від 1 до *(p-l)* включно; обчислення провадяться за модулем *p,* де *p*- велике просте число; *a* - ціле число ($1 \leq a \leq p$).

Якщо $y = a^x$, то природно записати:

$$x = log_a(y),$$

а задачу обернення f(x) назвати задачею знаходження дискретних логарифмів. Навіть при дуже великих *p,* наприклад, $p \sim 2^{1000}$ можна легко обчислити *f(x)* піднесенням до квадрату та множенням. Наприклад, для того щоб обчислити $a^{53} = a^{32+16+4+1}$ потрібно спочатку знайти $a^2, a^4 = (a^2)^2, a^8 = (a^4)^2, a^{16} = (a^8)^2$; для цього потрібно 5 операцій множення. Далі слід помножити послідовно a^{32} на a^{16}, a^4 і a, що потребує ще трьох операцій. Отже, результат дістають за вісім операцій (за модулем *p*). Навіть при $p \sim 2^{1000}$ для обчислення *f(x)* для довільного цілого числа *x* потрібно менше ніж 2000 операцій множення (за модулем *p*).

Якщо функція дискретного піднесення до степеня дійсно одностороння, то обчислення $log_x y$ має бути нездійсненним практично для всіх y ($1 \leq y < p$). Невдовзі М.Е.Хеллман і С.Поліг з'ясували, що дискретні логарифми складно

обчислити за умови, коли не тільки p велике, але і p-1 має великий простий множник (найкраще всього, якщо це друге просте число, помножене на 2). За цієї додаткової умови кращі відомі алгоритми для знаходження дискретних логарифмів потребують приблизно $\sqrt{p}$множень (за модулем *p)* у порівнянні з приблизно $2log_2 p$ множень при дискретному піднесенні до степеня.

Діффі і Хеллман запропонували простий метод використання дискретних логарифмів для обміну секретними ключами між користувачами мережі з використанням лише відкритих повідомлень.

Припустимо, що всім користувачам відомі a і p. Кожен користувач, скажімо, користувач *i*, випадково вибирає ціле число x_i, яке знаходиться між 1 і p-1, і тримає його в секреті. Далі він обчислює y_i:

$$y_i = a^{x_i} \, mod \, p.$$

Користувач не тримає y_i в секреті, а розміщує його в завірений відкритий довідник, доступний для всіх користувачів. Надалі, якщо користувачі i та j побажають встановити секретний зв'язок, користувач *i* візьме із довідника y_i і з допомогою свого секретного x_i, обчислить Z_{ij}:

$$Z_{ij} = (y_j)^{x_i} = (a^{x_j})^{x_i} = a^{x_j x_i} \, mod \, p.$$

У такий самий спосіб i користувач j обчислить Z_{ji}. Однак $Z_{ji} = Z_{ij}$, і користувачі i та j можуть з цього моменту використовувати Z_{ij}, як секретний ключ у класичній криптосистемі. Якщо зловмисник зміг би розв'язати задачу обчислення дискретних логарифмів, то зміг би за відомими з довідника y_i і y_j розв'язати рівняння $x_i = log_a(y_i)$ і обчислити Z_{ij}, як і користувач *i*. Напевно, зловмисник не може визначити Z_{ij} іншим шляхом (хоч це і не доведено). Схема, яку ми описали, дістала назву *системи відкритого розповсюдження ключів Діффі і Хеллмана.*

Це перша система, яка дає можливість відмовитися від передачі секретних ключів. На сьогодні її вважають однією з найстійкіших і найзручніших систем з відкритими ключами.

Зазначимо, що описана система відкритого розповсюдження ключів дає змогу обійтися без захищеного каналу для передачі секретних ключів, але не відміняє необхідності автентифікації. Держатель загальнодоступного довідника повинен бути впевненим, що несекретне y_i, помістив у довіднику саме користувач i, а користувач i — мати впевненість, що y_i надіслав йому саме держатель довідника. Не треба забувати і про те, що у системах з секретними ключами користувач повинен бути впевненим не тільки в тому, що ключ Z зберігався в секреті під час передавання, а й у тому, що він надісланий справжнім відправником. Методи з відкритими ключами усувають одну з цих проблем. Вони не створюють нової задачі автентифікації, а, скоріше, акцентують на необхідність її розв'язання.

Схема розповсюдження ключів показана на рисунку 4.1.

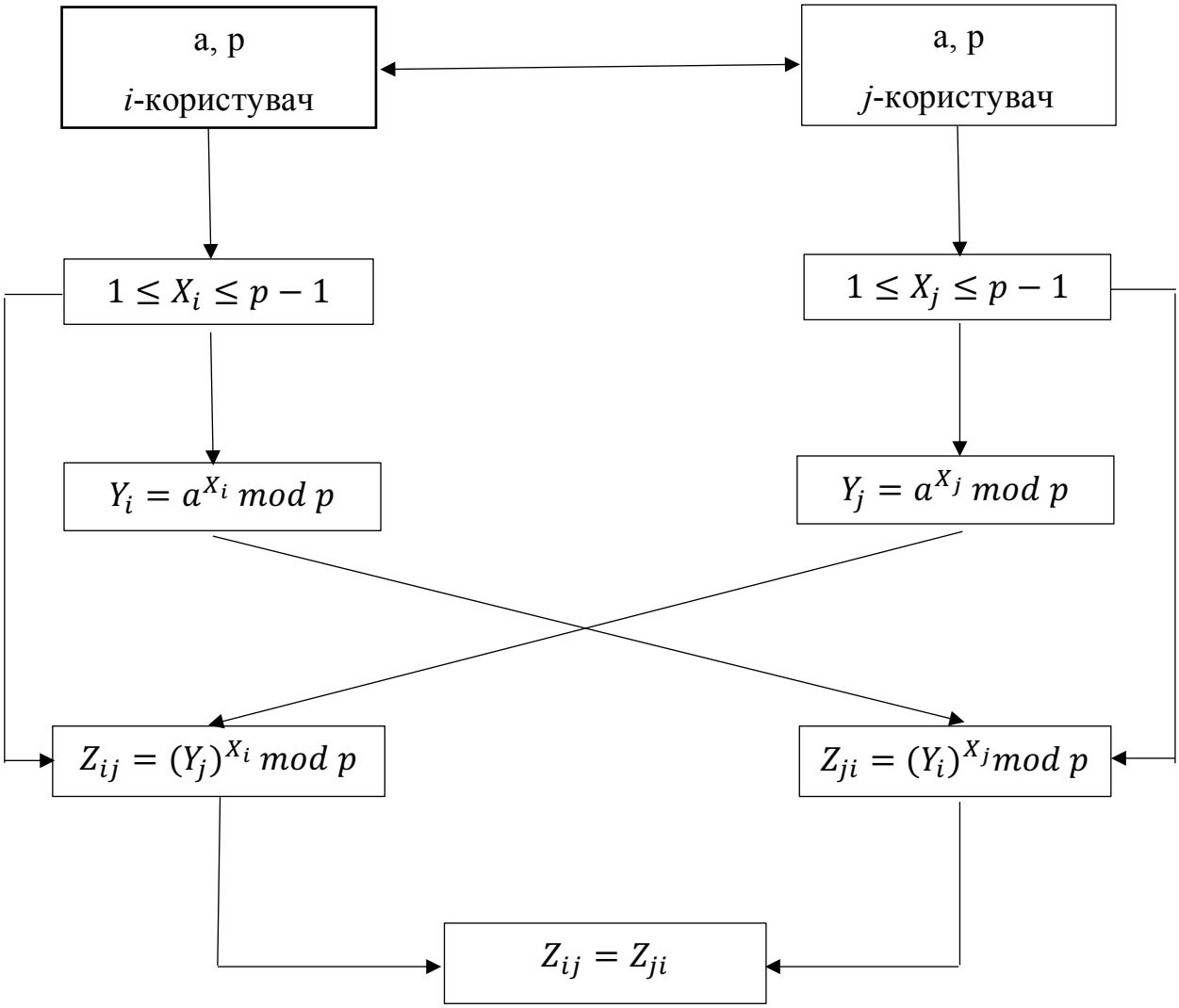

Рисунок 4.1 – Схема розповсюдження ключів

Відмітимо, що вибір *a* і *p* може суттєво впливати на криптостійкість. Зауважимо також, що *(p* - 1)/2 повинно бути простими.

Без додаткових застережливих заходів (введення сертифікатів відкритих ключів) описаний метод ключового обміну в разливий з точки зору атаки, відомої під назвою “людина посередині” (man-in-the-middle attack).

Припустимо, зловмисник може не лише підслуховувати повідомлення абонентів *i* та у, але і змінювати повідомлення, а також створювати зовсім нові фальшиві повідомлення. Тоді зловмисник може видавати себе за *j*, який сповіщає дещо *i,* або за *i* який повідомляє дещо *j*. Атака складається з таких дій:

1) абонент *i* посилає абоненту *j* свій відкритий ключ, зловмисник перехоплює його і посилає у свій власний відкрити; ключ;

2) *j* посилає *i* свій відкритий ключ, зловмисник перехоплює його і посилає *i* свій власний відкритий ключ;

3) коли *i* посилає повідомлення у, яке зашифроване на його (*j*) відкритому ключі, зловмисник його перехоплює. Так я повідомлення в дійсності зашифроване на відкритому ключі зловмисника, він розшифровує його, знову зашифровує його на відкритому ключі *j* і посилає *j*;

4) коли *j* посилає повідомлення *i*, яке зашифроване на його *(i)* відкритому ключі, зловмисник його перехоплює. Так я повідомлення в дійсності зашифроване на відкритому ключі зловмисника, він розшифровує його і потім знову зашифровує на відкритому ключі *i* та відсилає *i.*

Атака можлива, навіть якщо відкриті ключі *i* та *j* зберігаються в базі даних. Зловмисник може перехопити запит і до бази даних та підмінити відкритий ключ у своїм власним. Те ж саме він може зробити з відкритим ключем *i*. Зловмисник може атакувати базу даних і підмінити ключі *i* та *j* своїми власними. Потім, дочекавшись, коли *i* та *j* почнуть обмінюватись повідомленнями, виконає перехоплення і підміну. Така атака досить ефективна, так як у *i* та j немає можливості перевірити, чи дійсно вони спілкуються один з одним. Якщо втручання зловмисника не приводить до помітних затримок при передачі повідомлень, абоненти не зможуть запідозрити, що хтось, розташований між ними, читає їхні секретні повідомлення.

Відкриті ключі повинні пройти “сертифікацію”, щоб попередити атаки, які пов’язані з підміною ключів (в першу чергу для протидії атаці “людина посередині”), і повинні регулярно змінюватись.

Протокол ключового обміну для кількох учасників. Описаний вище алгоритм можна легко розширити для випадку трьох і більше учасників. У наведеному нижче прикладі *i, j, k* разом генерують загальний секретний ключ.

Крок 1. *i* вибирає велике випадкове ціле число x_i і обчислює

$$y_i = a^{x_i} \, mod \, p.$$

Крок 2. *j* вибирає велике випадкове ціле число x_j і надсилає k

$$y_j = a^{x_j} \, mod \, p.$$

Крок 3. *k* вибирає велике випадкове ціле число x_k і надсилає *i*

$$y_k = a^{x_k} \, mod \, p.$$

Крок 4. *i* надислає *j*

$$Y_k = y_k^{x_i} \, mod \, p$$

Крок 5. *j* надислає *k*

$$Y_i = y_i^{x_j} \, mod \, p$$

Крок 6. *k* надислає *i*

$$Y_j = y_j^{x_k} \, mod \, p$$

Крок 7. *i* обчислює

$$k1 = Y_j^{x_i} \, mod \, p$$

Крок 8. *j* обчислює

$$k1 = Y_k^{x_j} \, mod \, p$$

Крок 9. *k* обчислює

$$k1 = Y_i^{x_k} \, mod \, p$$

Секретний ключ $k1$ дорівнює $a^{x_i x_j x_k} \, mod \, p$, і ніхто з зловмисників, що підслуховують канал, не зможе його обчислити. Протокол можна легко розширити для чотирьох і більше учасників.

Одностороння генерація ключа. Ця модифікація методу Діффі-Хеллмана дозволяє абоненту *i* згенерувати ключ і надіслати його абоненту j.

Крок 1. Абонент *i* вибирає велике випадкове ціле число x_i, та генерує

$$k = a^{x_i} \, mod \, p$$

Крок 2. Абонент *j* вибирає велике випадкове ціле число x_j, та посилає абоненту *i*

$$y_j = a^{x_j} \, mod \, p$$

Крок 3. Абонент *i* посилає абоненту *j*

$$x = y_j^{x_i} \, mod \, p$$

Крок 4. Абонент *j* обчислює

$$z = x_j^{-1},$$

$$k' = x^z \ mod \ p$$

Якщо все виконано правильно; то$k' = k$. Перевага цього методу полягає у тому, що k можна обчислити заздалегідь, до взаємодії, і абонент i може шифрувати повідомлення за допомогою k задовго до встановлення сполучення з абонентом j. Абонент може послати повідомлення відразу кільком абонентам, а передати пізніше - кожному окремо.

4.1.1 Проведення експериментів

Програмний продукт був реалізований у вигляді бібліотеки з методами піднесення до степення за модулем, що були представленні в попередніх пунктах, а також алгоритмом Діффі-Хеллмана з використанням різних методів піднесення до степення за модулем.

Експеримент проводився на обладнанні з наступними характеристиками:

- процесор з тактовою частотою 2.2 ГГц;
- об’єм оперативної пам’яті 16 ГБ;
- операційна система macOS High Sierra.

Результати експерименту наведені в таблиці 4.1 (в таблиці показано, які алгоритми були взяті за основу піднесення до степення за модулем).

Таблиця 4.1 – Результати експериментів

	Алгоритм “зліва направо”	m-арний алгоритм		CLNW	
k	*T*, мс	*r*	*T*, мс	*r*	*T*, мс
128	9128	4	7504	4	7128
256	18372	4	15264	5	14524
512	38764	5	30428	5	29824
768	59912	5	48632	6	44496
1024	73464	5	58152	6	55396
2048	84188	6	64836	7	60840

Результати експерименту також наведені на рисунку 4.2.

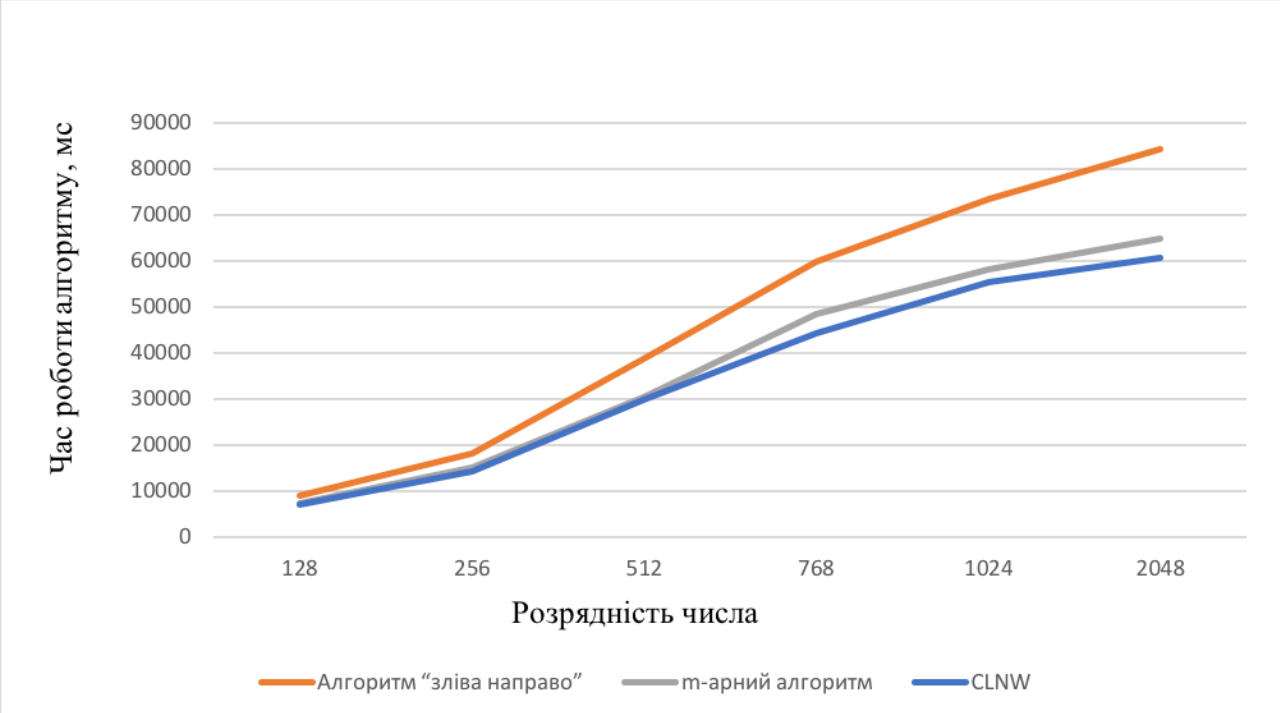

Рисунок 4.2 – Результати експериментів для алгоритму Діффі-Хеллмана

4.2 Висновки до розділу

В даному розділі було описано схему розповсюдження ключів Діффі-Хеллмана, а також проведено експерименти, щодо роботи алгоритму Діффі-Хеллмана з різними алгоритмами піднесення до степення за модулем.

Найкращий час вдалося отримати з адаптивним m-арним алгоритмом Ненульових вікон однакої довжини (CLNW).

5 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для ілюстрації роботи швидких алгоритмів обчислення криптопримітивів та їх застосування для алгоритмів криптографії було прийнято рішення розробити власну бібліотеку з реалізованими алгоритмами, що були описані вище, використавши інструменти, що зарекомендували себе для обробки вхідних даних, візуалізації результатів роботи алгоритмів та зручного API для інтеграції в інші системи. Це дозволяє зосередитися над розробкою самих алгоритмів та швидкої оцінки результатів їх роботи.

5.1 Засоби розробки

Наш програмний продукт це бібліотека з набором інструментів, API, що можуть бути інтегровані в інші системи. Швидкі алгоритми обчселення криптопримітивів були написані на мові Python. Так як ця мова програмування зарекомендувала себе як одна з найбільш популярних, ефективних для вирішення задач, пов’язаних з розробкою алгоритмів машинного навчання та аналізу даних. Для розробки, тестування та візуалізації результатів було обрано Jupyter Notebook.

Jupyter Notebook – це веб застосування з відкритим висхідним кодом, що дозволяє створювати документи, що містить в собі виконуючий код, засоби для візуалізації даних, обробки даних. Дане застосування є потужним інструментом для дослідження поставлених задач.

Даний програмний продукт працює під операційними системами Windows, Linux, OS X. Є можливість вигрузити розроблені алгоритми в вигляді набору файлів для подальшого використання їх в інтеграції з іншими система, хмарними рішеннями та в якості десктопних застосунків на ПК.

5.2 Вимоги до технічного забезпечення

Для коректної роботи веб застосунка Jupyter Notebook необхідний сервер з наступними характеристиками:

а) процесор з тактовою частотою не нижче 2.6 ГГц;

b) достатній об’єм оперативної пам’яті не менше 4 ГБ;

c) операційна система Centos (7.4.1708).

Для коректної роботи бібліотеки потрібно встановити додаткові сторонні Python бібліотеки, що наведені в таблиці 5.1.

Таблиця 5.1 – Сторонні Python бібліотеки

Назва	**Версія**
numpy	1.14.2
scipy	1.0.0
pandas	0.20.3
attrdict	2.0.0
notebook	5.4.1

5.3 API програмного продукту

lib.multiply.standard.multiply (num1: int, num2: int) -> int

Множення двох чисел, використовуючи стандартний підхід.

Параметри:

– *num1* – число типу integer;

– *num2* – число типу integer.

Результат: число типу integer.

lib.multiply.fft.multiply (num1: int, num2: int) -> int

Множення двох чисел, використовуючи швидке перетворення Фур’є.

Параметри:

– *num1* – число типу integer;

– *num2* – число типу integer.

Результат: число типу integer.

lib.exp.left_right.exp(x: int, n: int, m: int) -> int

Піднесення числа до степення за модулем, використовуючи бінарний метод “зліва-направо”.

Параметри:

– *x* – число типу integer;

– *n* – степінь, число типу integer;

– *m* – модуль, число типу integer.

Результат: число типу integer.

lib.exp.quarternal.exp(x: int, n: int, m: int) -> int

Піднесення числа до степення за модулем, використовуючи m-арний алгоритм.

Параметри:

– *x* – число типу integer;

– *n* – степінь, число типу integer;

– *m* – модуль, число типу integer.

Результат: число типу integer.

lib.exp.clnw.exp(x: int, n: int, m: int) -> int

Піднесення числа до степення за модулем, використовуючи CLNW алгоритм.

Параметри:

– *x* – число типу integer;

– *n* – степінь, число типу integer;

– *m* – модуль, число типу integer.

Результат: число типу integer.

lib.exp.vlnw.exp(x: int, n: int, m: int) -> int

Піднесення числа до степення за модулем, використовуючи VLNW алгоритм.

Параметри:

– *x* – число типу integer;

– *n* – степінь, число типу integer;

– *m* – модуль, число типу integer.

Результат: число типу integer.

Приклад застосування описаних вище бібліотек показано на рисунку 5.1.

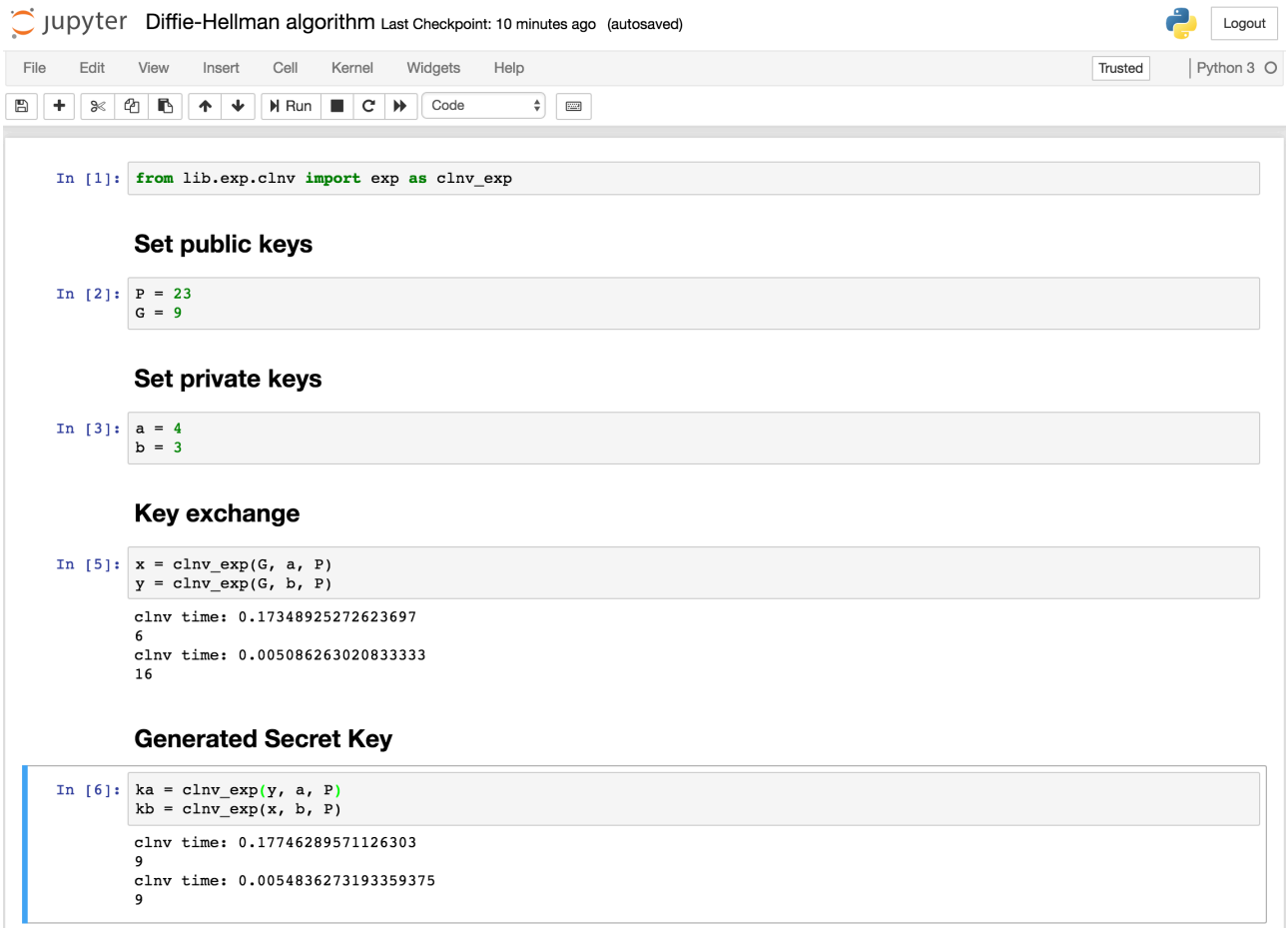

Рисунок 5.1 – Алгоритм Діффі-Хеллмана в середовищі Jupyter Notebook

5.4 Висновки до розділу

В цьому розділі зроблено детальний огляд засобів розробки та наведено обґрунтування їх вибору. Також наведені вимоги до технічного забезпечення,

необхідного для роботи програмного забезпечення. Описано API розробленої бібліотеки та показано приклад застосування.

ВИСНОВКИ

В результаті виконання роботи було виконано всі поставлені задачі:

– виконано огляд існуючих методів та засобів шифрування та дешифрування інформації;

– здійснено порівняльний аналіз алгоритмів множення цілих чисел;

– здійснено порівняльний аналіз алгоритмів піднесення числа до степення за модулем;

– модифіковано алгоритм відкритого розповсюдження ключів Діффі-Хеллмана з використанням швидких алгоритмів обчислення криптопримітивів.

– розроблено програмну реалізацію модифікованого алгоритму Діффі-Хеллмана;

– виконано аналіз отриманих результатів.

Теоретично та експериментально було показано, що підхід множення чисел з використанням алгоритму швидкого перетворення Фур'є працює швидше стандартного підходу множення двох чисел, для багаторозрядних чисел.

Операцію піднесення числа до степення за модулем для багаторозрядних чисел можна прискорити завдяки адаптивним m-арним алгоритмам.

Завдяки описаним вище підходам отримуємо резерв для оптимізації алгоритмів криптографії, що базуються в основному на операціях додавання, множення та піднесення до степення, що і було продемонстровано на основі алгортму відкритого розповсюдження ключів Діффі-Хеллмана.

ПЕРЕЛІК ПОСИЛАНЬ

1. Задірака В.К., Олексюк О.С. Комп'ютерна арифметика багаторозрядних чисел. //Київ - 2003
2. Карацуба А.А., Офман Ю.П. Умножение многорозрядных чисел на автоматах //ДАН СССР. — 1962. т.145. — С. 293-294.
3. Шенхаге А., Штрассен В. Быстрое умножение больших чисел // Кибернет. сб. — 1973. — вып. 10. — С. 87-98.
4. Cook S. A., Aanderaa S. O. On the minimum computation time of functions, Thesis, Harvard University, 1966. — P. 26-50.
5. Березовский А.И., Задирака В.К., Шевчук Л.Б. О тестировании быстродействия алгоритмов и программ вычисления основных операций ассиметричной криптографии /Кибернетика и системный анализ № 5, 1999. - С. 61-68.
6. Кнут Д.Е. Искусство программирования для ЭВМ. Т.2.- М.: Издательский дом "Вильямс", 2001. - 828 с.
7. Качко Е.Г., Свинарёв А.В., Горбенко И.Д., Мельникова О.А. Программирование операций многократной точности //Безопасность информации, №1,1995, с. 18-21.
8. Comba P.G. Exponention cryptosystems on the IBM PC //IBM Systems J. B 1990. - 29. - № 4. - P. - 526-538.
9. I Pollard J. M. The fast Fourier transform in a finite field. Mathematics of Computation, 25,1971.- P. 365 - 374.
10. Riesel H. Prime Numbers and Computer Methods for Factorization. Boston, MA: BirkMnser, 1985.
11. Egecioglu 0. and K09 Q. K. Exponentiation using canonical recording. Theoretical Computer Science, 129 (2), 1994. — P. 407 — 417.
12. U. Задирака В.К., Мельникова С.С. Цифровая обработка сигналов. - К.: Наукова думка, 1993. - 294с. 9
13. Lipson J. D. Elements of Algebra and Algebraic Computing. Reading, MA: Addison-Wesley, 1981.

14. Шеннон К.Э. Теория связи, в секретных системах. В работах по теории информации и кибернетики.-г М.:И.Л., 1963. Г
15. Brickel! E.F. Survey of Hardware Implementation of RSA; //Proc. of CRYPTO'89. Lecture Notes in Comp. Sci. Springer-Verlag, 1990, v. 435.-P. 368-370.
16. Lacy J.B., Mitchell D.P., Schell W.M. Cryptolib: Cryptography in Software. //Proc. of UNIX Security Symposium IV. USENIX Association, 1993-P. 1-
17. CCITT. Recommendation X.509: The Directory. - Authentication Framework. 1988.
18. Balenson D. Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes and Identifiers. //RFC 1423, Feb. 1993.
19. PKCS#1: RSA Encryption Standard. Version 2.0. RSA Data Security Inc. 1997.
20. ElGamal T. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. //Proc. of CRYPTO'84. Lecture Notes in Comp. Sci. Springer-Verlag, 1985, V.196.-P. 10-18.
21. Heilman V.S. Patent № 4.200.770,29 Apr 1980.
22. Bert den Boer. Diffie-Hellman is a strong as Discrete Log for certain primes. In CRYPTO'88 Proc. Springer-Verlag, 1990. Lecture Notes in Computer Science, v.403. - P. 530-539.
23. Brickell E. F. and McCarley K. S. An interactive identification scheme based on discrete logarithms and factoring, J. Cryptology, №5(1), 1992.-P. 29-39.
24. Chaum D. On line cast checks. //Proc. EUROCRYPT'89, Lecture j Notes in Comput. Sci., v.434, 1990.1P. 288-293.
25. Миренков Н. Н. Параллельное программирование для многомодульных вычислительных систем. - М.: 1989.-320 с; 10
26. Shor P. Algorithms for Quantum Computation: Discrete!! Logarithms and Foundations of Computer Science. IEEE Computer! Society Press, 1999. P. 124-134.

27. Boneh D., Lipton R. J. Quantum cryptanalysis of hidden linear я functions // Lect. Notes Comput. Sci № 963,1995. - P. 424-437

ДОДАТОК А Графічні матеріали

Плакат 1 Множення цілих чисел з використанням алгоритму швидкого перетворення Фур'є

Плакат 2 Порівняння часу роботи алгоритмів множення

Плакат 3 Порівняння часу роботи алгоритмів піднесення до степення за модулем

Плакат 4 Відкрите розповсюдження ключів Діффі-Хеллмана

Плакат 5 Час роботи алгоритму Діффі-Хеллмана з використанням різних методів піднесення до степення за модулем

Плакат 6 Алгоритм Діффі-Хеллмана в середовищі Jupyter Notebook